DataGRID

Final Evaluation Report for Data Management

Document identifier:	**DataGrid-02-D2.6**
EDMS id:	**http://edms.cern.ch/document/406379/**
Date:	December 11, 2003
Work package:	**WP2: Data Management**
Partner(s):	**CERN, PPARC, HIP, INFN, ITC, KDC**
Lead Partner:	**CERN**
Document status:	**UNDER PTB REVIEW**
Author(s):	WP2
File:	**finalReport**

Abstract: This document contains an overview of the final work done within the Data Management Work-package throughout the entire scope of the project.

CONTENTS

Delivery Slip

.	Name	Partner	Date	Signature
From	WP2	CERN, PPARC, HIP, INFN, ITC, KDC		
Reviewed by	Frederic Hemmer David Groep Antony Wilson	CERN NIKHEF PPARC		
Approved by	PTB			

Document Change Record

Issue	Date	Comment	Author
0_1	3 Oct 2003	First draft	Heinz Stockinger
0_2	7 Oct 2003	Additions to the Chapter 2	Heinz Stockinger
0_3	8 Oct 2003	Added basic structure for evaluation of tools	Heinz Stockinger
0_4	17 Oct 2003	Added draft of Sec 3.4 about optor-sim	Floriano Zini
0_5	22 Oct 2003	Updated optimisation and evaluation sections.	Kurt Stockinger
0_6	27 Oct 2003	Expanded architecture section, proofreading	Leanne Guy
0_7	29 Oct 2003	Revised section on Optorsim and made it consistent	Floriano Zini
0_8	30 Oct 2003	Moved OptorSim in separate section + updates	Kurt Stockinger
0_9	31 Oct 2003	Proofreading of OptorSim section	Caitriana Nicholson
0_9_1	5 Nov 2003	More information added on RLS performance and Replica Manager	Heinz Stockinger
0_9_2	11 Nov 2003	Rewrote a lot of the architecture replication section	Leanne Guy
		Factored out a common web-services section from the Spitfire section	
0_9_3	13 Nov 2003	1.8 TB transfer results added	Heinz Stockinger
0_9_4	13 Nov 2003	Added sections on metadata, common components and optimization work	Gavin McCance
0_10	17 Nov 2003	Expanded the last chapter on open issues and future work.	Peter Kunszt
0_11	20 Nov 2003	Overall review of full document.	Peter Kunszt
1_0	24 Nov 2003	Minor final edits	Heinz Stockinger
1_1	1 Dec 2003	Comments from David Groep and Antony Wilson included	Heinz Stockinger
1_2	5 Dec 2003	Comments from Frederic Hemmer included	Peter Kunszt
1_3	5 Dec 2003	Rework the exec summary	Peter Kunszt

1. INTRODUCTION

This document provides the final report of the Data Management work package (Work Package 2, WP2) of the EU DataGrid project. We present an overview and evaluation of all the delivered data management components and services. Furthermore we discuss added data management components that formed part of the original architectural design but that could not be implemented within the time constraints of the EDG project. We discuss the additional functionality that they would bring to the existing data management services and how they might be implemented in future projects.

1.1. APPLICATION AREA

This document applies to Grid Data Management and reports on the work-plan presented in the EU DataGrid Technical Annex [1] and the Data Management Design document [3].

1.2. DOCUMENT STRUCTURE

Before we go into the detailed discussion of all achievements, the **goals** of the work-package are summarised in section 3. This section also gives a short overview of the main software tools provided within the lifetime of the project. The data management tasks are divided into the following sub-tasks where several results in terms of software and publications have been gained:

- Data Access and Migration
- Replication
- Metadata Management
- Security and Transparent Access
- Query Optimisation Support and Access Pattern Management

A detailed overview of the second generation architecture is given in section 4. The main components of the Data Management architecture are:

- Replica Manager
- Replica Location Service incl. Replica Metadata Catalog
- Replica Optimisation Service
- Security Components
- Database Access Service

All these components have been included and integrated into the latest EDG software release. Detailed **performance and scalability results** of each of the services and components can be found in section 5.

Since Data Grids generally provide a very complex infrastructure, we have built a **Grid simulator called OptorSim** (see section 6.) in order to study the behaviour and interactions of several Grid services under particular load. The results of the simulations have been directly used for the developed services and in particular for the Replica Optimisation Service.

Conclusions and a detailed summary of open issues and future work are given in section 7.

1.3. Reference Documents

References

[1] EU DataGrid Technical Annex, IST-2000-25182, 24 August 2000. http://cern.ch/eu-datagrid/EUDocuments/TechnicalAnnex/Technical%20Annex.htm

[2] D2.1 Report on Current Technology: Data Access and Mass Storage, EDG Deliverable 2.1, 20 December 2001. http://cern.ch/edg-wp2/docs/DataGrid-02-D2.1-0105-2_0.pdf

[3] Wolfgang Hoschek, Javier Jaen- Martinez, Peter Kunszt, Ben Segal, Heinz Stockinger, Kurt Stockinger, Brian Tierney, "Data Management (WP2) Architecture Report", EDG Deliverable 2.2, http://edms.cern.ch/document/332390

[4] Data Management Workpackage, EU DataGrid: D2.2.A1 Addendum to the Data Management Architecture Report (Covering Testbed Release 2.0), EDG Deliverable 2.2.A1 http://edms.cern.ch/document/374107/addendum.pdf

[5] Data Management Workpackage, EU DataGrid: D2.5 Components and Documentation for the Final Project Release. https://edms.cern.ch/file/407063/1/finalDocumentationWP2.pdf

[6] DataGrid WP1, Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description, *Technical Report, EU DataGrid Project. Deliverable D1.2,* September 2001. https://edms.cern.ch/file/332413/1/datagrid-01-d1.2-0112-0-3.pdf

[7] Wolfgang Hoschek, Javier Jean-Martinez, Asad Samar, Heinz Stockinger, Kurt Stockinger. Data Management in an International Data Grid Project. *1st IEEE/ACM International Workshop on Grid Computing (Grid'2000)*. Bangalore, India, Dec 17-20, 2000.

[8] W. Allcock, J. Bester, J. Bresnahan, A. Chernevak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke; "Data Management and Transfer in High Performance Computational Grid Environments." Parallel Computing, 2002. http://www.globus.org/research/papers/dataMgmt.pdf

[9] CASTOR web-site: http://cern.ch/castor

[10] Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane: "A Java Commodity Grid Kit", Concurrency and Computation: Practice and Experience, 13(8-9), 2001. http://www.globus.org/research/papers/vonLaszewski–cog-cpe-final.pdf

[11] Heinz Stockinger, Flavia Donno, Erwin Laure, Shahzad Muzaffar, Peter Kunszt, Giuseppe Andronico, and Paul Millar. Grid Data Management in Action: Experience in Running and Supporting Data Management Services in the EU DataGrid Project. In Computing in High Energy Physics (CHEP 2003), La Jolla, California, March 24-28 2003.

[12] Diana Bosio, James Casey, Akos Frohner, Leanne Guy, Peter Kunszt, Erwin Laure, Sophie Lemaitre, Levi Lucio, Heinz Stockinger, Kurt Stockinger, William Bell, David Cameron, Gavin McCance, Paul Millar, Joni Hahkala, Niklas Karlsson, Ville Nenonen, Mika Silander, Olle Mulmo, Gian-Luca Volpato, Giuseppe Andronico, Federico DiCarlo, Livio Salconi, Andrea Domenici, Ruben Carvajal-Schiaffino, and Floriano Zini.Next-Generation EU DataGrid Data Management Services. In Computing in High Energy Physics (CHEP 2003) , La Jolla, California, March 24-28 2003.

[13] Heinz Stockinger, Asad Samar, Shahzad Muzaffar, and Flavia Donno. Grid Data Mirroring Package (GDMP). Scientific Programming Journal - Special Issue: Grid Computing, 10(2):121-134, 2002.

[14] Heinz Stockinger, Asad Samar, Bill Allcock, Ian Foster, Koen Holtman, and Brian Tierney. File and Object Replication in Data Grids. Journal of Cluster Computing, 5(3):305-314, 2002.

[15] Andrea Domenici, Flavia Donno, Gianni Pucciani, Heinz Stockinger, Kurt Stockinger. Replica Consistency in a Data Grid. IX International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT03), Tsukuba, Japan, Dec. 1-5, 2003.

[16] Particle Physics Data Grid (PPDG): http://www.ppdg.net

[17] L. Guy, P. Kunszt, E. Laure, H. Stockinger, K. Stockinger "Replica Management in Data Grids", Technical Report, GGF5 Working Draft, Edinburgh Scotland, July 2002

[18] Peter Kunszt, Erwin Laure, Heinz Stockinger, and Kurt Stockinger. Advanced Replica Management with Reptor . In 5th International Conference on Parallel Processing and Applied Mathematics, Czestochowa, Poland, September 7-10, 2003. Springer Verlag.

[19] Ann Chervenak, Ewa Deelman, Ian Foster, Leanne Guy, Wolfgang Hoschek, Adriana Iamnitchi, Carl Kesselman, Peter Kunszt, Matei Ripeanu, Bob Schwartzkopf, Heinz Stockinger,Kurt Stockinger, and Brian Tierney. "Giggle: A Framework for Constructing Scalable Replica Location Services." In Proc. of the International IEEE Supercomputing Conference (SC 2002), Baltimore, USA, November 2002.

[20] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, F. Zini, "Design of a Replica Optimisation Framework", DataGrid-02-TED-021215

[21] Amy Krause, Susan Malaika, Gavin McCance, James Magowan, Norman W. Paton, Greg Riccardi "Grid Database Service Specification", Global Grid Forum 6, Edinburgh, 2002.

[22] R. Housley et.al. "Internet X.509 Public Key Infrastructure Internet X.509 Public Key Infrastructure, RFC 3280, The Internet Society April 2002, http://www.ietf.org/rfc/rfc3280.txt

[23] W3C. "Web Services Activity", http://www.w3c.org/2002/ws/

[24] http://www.w3.org/TR/2003/WD-ws-gloss-20030808/

[25] William Bell, Diana Bosio, Wolfgang Hoschek, Peter Kunszt, Gavin McCance, and Mika Silander. "Project Spitfire - Towards Grid Web Service Databases". Technical report, Global Grid Forum Informational Document, GGF5, Edinburgh, Scotland, July 2002.

[26] Kurt Stockinger, Heinz Stockinger, Lukasz Dutka, Renata Slota, Darin Nikolow, and Jacek Kitowski. Access Cost Estimation for Unified Grid Storage Systems. In 4th International Workshop on Grid Computing (Grid2003), Phoenix, Arizona, November 17, 2003. IEEE Computer Society Press.

[27] Apache Axis is an implementation of the SOAP ("Simple Object Access Protocol") submission to W3C. http://ws.apache.org/axis/

[28] Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. http://jakarta.apache.org/tomcat/

[29] The Jakarta Project creates and maintains open source solutions on the Java platform for distribution to the public at no charge. http://jakarta.apache.org/

[30] Web Service Definition Language, http://www.w3.org/TR/wsdl

[31] Java Naming and Directory Interface, http://java.sun.com/products/jndi/

[32] Open Grid Services Infrastructure, http://www.gridforum.org/ogsi-wg/

[33] Open Grid Services Architecture, http://www.gridforum.org/ogsa-wg/

[34] Storage Resource Management (SRM) working group: http://sdm.lbl.gov/srm-wg/

[35] GRIDSTART web-site: http://www.gridstart.org

[36] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *Int. Journal of High Performance Computing Applications*, 17(4), 2003.

[37] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, P. Millar, K. Stockinger, and F. Zini. Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In *Int. Workshop on Agent Based Cluster and Grid Computing at CCGrid2003*, Tokyo, Japan, May 2003. IEEE-CS Press.

[38] David G. Cameron, Ruben Carvajal-Schiaffino, A. Paul Millar, Caitriana Nicholson, Kurt Stockinger, Floriano Zini, Evaluating Scheduling and Replica Optimisation Strategies in OptorSim, International Workshop on Grid Computing (Grid2003), Phoenix, Arizona, November 2003, IEEE-CS Press.

[39] G. K. Zipf, Selected Studies of the Principle of Relative Frequency in Language, Cambridge, MA. Harvard University Press, 1932.

[40] Local Replica Catalog Installation Guide, http://cern.ch/edg-wp2/replication/docu/r2.1/edg-lrc-installguide.pdf

1.4. TERMINOLOGY

Endpoint	**Web Services Endpoint**	An association between a service binding and a network address, specified by a URI, that may be used to communicate with an instance of a service. It indicates a specific location for accessing a service using a specific protocol and data format. [24]
GDMP	**Grid Data Mirroring Package**	A Grid tool for data mirroring/replication.
GSI	**Grid Security Infrastructure**	Globus Security mechanism
GUID	**Grid Unique IDentifier**	A UUID generated by the Replica Management System for an SURL. A GUID is created every time a new SURL is registered and is always immutable.
LFN	**Logical File Name**	A Logical File Name is a user defined alias to a GUID. Unlike GUIDs, aliases are mutable but should still be globally unique. Since the Replica Management System has only limited control over the creation of LFNs, this global uniqueness is only weakly enforced.
LRC	**Local Replica Catalog**	A catalog that stores GUID to SURL mappings together with user defined SURL attributes for data files stored locally.
RLI	**Replica Location Index**	An RLI indexes over all LRCs that subscribe to it. The RLI stores GUID to LRC mappings, thus maintaining an index of all LRCs that store at least one physical replica of a given GUID. RLIs provide a link between many LRCs in a grid hence providing a distributed index and querying mechanism over all LRCs.
RLS	**Replica Location Service**	The RLS is a system that maintains information about the physical locations of logical identifiers of data and provides access to this information. It comprises two internal components, an LRC and an RLI and so provides alias to physical file mappings for all replicas located on the Grid for a given Virtual Organization. It is the successor to the Globus Replica Catalog used in EDG releases earlier than 2.0.
RMC	**Replication Metadata Catalogue**	The Replication Metadata Catalog is a system that maintains LFN to GUID mappings, as well as attributes on GUIDs and LFNs.
RPC	**Remote Procedure Call**	RPC is the synchronous transfer of control mechanism between programs in disjoint address-spaces when the communication medium is the network.
SE	**Storage Element**	A Grid Service where files can be stored and registered with a catalog.
SRM	**Storage Resource Manager**	A Storage Resource Manager is responsible for managing various kinds of storage devices and media such as a single disk, a disk pool (farm), a hierarchical storage system or a tape system. An SRM provides a unique interface to all these storage resources.

SFN	**Storage File Name**	An SFN is a locater for a physical file, where the scheme specific part is understood by a Storage Resource Manager (SRM). It is a URL where the scheme is ‘sfn’ and the host is a valid SRM host.
SURL	**Storage URL**	An SURL is a locator for a physical file, where the scheme specific part is understood by a Storage Resource Manager (SRM). It is a URL where the scheme is ‘srm’ and the host is a valid SRM host.
TURL	**Transport URL**	A Transport URL is returned by a SRM in response to a request for a way to access a SURL. It includes the actual protocol by which you can access the SURL by. For instance, ‘gsiftp’ for GridFTP, or ‘http’ for HTTP access.
UUID	**Universally Unique IDentifier**	A UUID is a 128 bit number that is guaranteed to be different from all other UUIDs (generated before 3400 A.D.) by the UUID standard upon which the implementation is based.
VO	**Virtual Organization**	Every user needs to be part of a community (or organisation). Since this organisation does not necessarily need to exist as a legal entity, it is called a “virtual organisation”. E.g. All people taking part in an EDG tutorial are part of the VO “tutor”.

2. Executive Summary

Work Package 2 has developed and deployed the necessary middleware to securely manage massive amounts of distributed data in a universal global name space. The functionality of the data management middleware delivered includes: movement and replication of data at high speed from one geographical site to another, management of remote copies data, optimization of access to data, and the provision of a metadata management tool. [1].

WP2 has contributed substantially to the establishment of international relationships with leading projects in the Grid community. A lot of the prototyping work carried out in WP2 has led to scientific publications.

Within the last three years, WP2 applied a prototyping software development approach with a first and a second generation of software tools. Whereas the first generation was mainly C++ based (GDMP, edg-replica-manager), the second generation is mainly implemented in Java and uses the web service paradigm. The experience with the early and first generations of middleware could be worked into the second generation tools to produce a stable and robust set of grid services. We have acquired a substantial knowledge of the web service technology and its limitations in the process.

The evaluation of the performance of these services shows that the required scaling can be achieved in a grid environment and that the response times are adequate. Of course there is room for a lot of improvement. In their current state, the replication services of WP2 are not complete as we have proposed them according to our initial design. The reason for this is the refocussing of the project strategy towards production-quality services, i.e. on robustness and stability as opposed to more functionality. Directions for future work include the implementation of the omitted services as well as the improvement in the performance and functionality of the existing services.

3. GENERAL GOALS AND WORK PACKAGE OVERVIEW

Work Package 2 had the task to provide the grid middleware for data management. This includes data movement, replication, access optimization and metadata management.[1].

The following section gives an overview of the responsibilities of each of the Data Management tasks as stated in the Technical Annex [1]. Architectural details and an evaluation can be found in subsequent sections.

It finishes with details on the management of WP2.

3.1. TASK 2.1: REQUIREMENTS DEFINITION

In the early phases of the project, requirements were collected from both the middleware and application work packages. This resulted in several requirements documents that were taken into account when designing the overall data management architecture, described in section 4. This work formed the basis for the two design documents [3] and [4].

3.2. TASK 2.2: DATA ACCESS AND MIGRATION

During the project, the “Data Access and Migration” task was merged with the “Replication” task, section 3.3., in order to provide transparent access to mass storage systems. An overview of existing mass storage solutions can be found in the main deliverable of that task [2].

3.3. TASK 2.3: REPLICATION

This task became the major task of the work package and thus resulted in the core of the data management services. The basic aim of replicating and locating files was demonstrated in several software systems produced by WP2. We classify these systems as either first or second generation services. The first generation services [11], such as GDMP (see below) and the edg-replica-manager, were essentially prototypes that provided a lot of useful feedback from the user communities. The second generation services were then designed based on the lessons learnt from the deployment and use in production of the first generation services. In Section 4. details about the second generation services are given together with brief statements about the first generation services.

3.3.1. GDMP - GRID DATA MIRRORING PACKAGE

GDMP (Grid Data Mirroring Package) [13, 14] is a service for the replication (mirroring) of file sets between several Storage Elements. The system was widely used by several High Energy Physics collaborations in Europe and the US for data mirroring. GDMP was developed and deployed in the first phases of the EDG project in collaboration with the Particle Physics Data Grid (PPDG) [16]. GDMP also provides a simple interface to Mass Storage Systems (MSS).

Together with the edg-replica-manager (section 3.3.2.), GDMP was deployed on the first and the second EDG testbeds up to release 1.4. Its use provided the main basis for the architectural design of the second generation replication service, presented in section 4.2.

3.3.2. EDG-REPLICA-MANAGER

The edg-replica-manager [11] was developed in the second and third year of the DataGrid project. It provides some added replication functionality that meets additional user requires that were identified during the use of GDMP in the EDG testbed. In this way, both tools (GDMP and edg-replica-manager) complemented each other and provided the basic replication functionality of the first generation replication tools.

3.4. Task 2.4: Metadata Management

Metadata appears in several forms within a Data Grid and thus is also addressed by several components. A short summary is given below and details will be provided in section 4.

- *Metadata for locating replicas*. This is generally regarded as the Replica Catalog problem and is covered by the Replica Location Service (see Section 4.2.2.).
- *General logical file specific metadata* such as file size (i.e. attributes of logical files), creation time etc. is stored in the Replica Metadata Catalog (see Section 4.2.3.).
- *General metadata* of any kind can be managed via the Spitfire tool as described in Section 4.3.

3.5. Task 2.5: Security and Transparent Access

The main achievements of this task are the Java based authentication and authorisation tools that are used for all the Java based web services provided by WP2 as well as the Java based web-services of WP3 (nformation and Monitoring Services) and WP5 (Mass Storage Management). The entire security infrastructure complies with the existing de-facto standard proposed by the Globus Alliance. More detail is given on the security layer in Section 4.4..

3.6. Task 2.6: Query Optimisation Support and Access Pattern Management

The field of query optimisation in a Grid required lots of research since it is a very new field in the Grid community. Early results were obtained with a simulation tool called OptorSim (developed by WP2) that is explained in more detail in section 6. The simulation permits the study of dynamic replica selection using economic models and measured access patterns.

The simulation results were then used to design and build the Grid Replica Optimisation Service described in Sections 4.2.4. and 5.4. As a result of the collaboration with the EU CrossGrid project, the replica optimization component was interfaced with their Data Access Estimator (DAES) [26].

3.7. Task 2.7: Testing, Refinement and Co-ordination

Software testing and refinement have been constant activities throughout the lifetime of the project. Testing tasks included test plans for each software package (see Deliverable 2.5 [5] for further details on test plans), software tests (unit tests, functional test suites, etc) and user requirements testing based on the constant feedback from the EDG users of the software deployed on the testbed.

The co-ordination of all activities in the work package was done in the following way:

- overall work package co-ordination: work package manager and deputy;
- co-ordination of specific tasks by task managers for: replication, metadata, security and optimisation;
- consulting with WP2 representatives in the following groups of the project: Integration Team, Security Group, Architecture Task Force, Quality Group, Tutorial Team.

3.8. INTERNATIONAL COLLABORATION

One of the major tasks of the EDG project was to establish international relationships with leading projects in the Grid community. Below is a list of the most outstanding and collaborations of WP2:

3.8.1. CO-OPERATION WITH GLOBUS

Since the first year of the project, EDG and in particular WP2 has had a close co-operation with the Globus Alliance on data management tools. This started with design reviews of the LDAP based replica catalogue and replica manager that were part of the Globus Toolkit 2. User feedback on the usage of the LDAP based data management services on the EDG testbed revealed several shortcomings in their design, which provided the basis for a redesign and implementation of the catalogue system. The result was an intensive collaboration on the Replica Location Service architecture [19] now implemented in Globus Toolkit 3 and EDG 2.x.

Unfortunately the two implementations, although semantically compatible, have incompatible (i.e. non-interchangeble) APIs. This is due to the fact that we have evolved our implementation according to the immediate needs of our community and did not have the necessary time to agree on a common interface. This deficiency will not be remedied for the time being as there is an effort to standardize on interfaces at GGF through the OREP (OGSA Replication) Working Group. Once this standard is defined, both implementations will be able to provide the necessary interface and interoperate.

3.8.2. STORAGE RESOURCE MANAGER

As part of Task 2.2, WP2 has contributed significantly to the specification of the Storage Resource Management (SRM) [34] standard for unified access to mass storage systems. This is an international collaboration involving many Grid projects, including the EDG and the Particle Physics Data Grid project (PPDG) in the U.S.

3.8.3. CO-OPERATION WITH THE EU CROSSGRID PROJECT

As part of the Optimisation Task 2.6, WP2 established a collaboration with the EU CrossGrid project on Storage Access Cost Estimation, which is documented in detail in [26]. The main goal was to make EDG's and CrossGrid's software systems inter-operable and use the access estimator for the replica selection process; both goals have been successfully attained.

3.8.4. GRIDSTART

One of the main aims of the GRIDSTART initiative [35] (an umbrella project that covers all FP5 and FP6 framework projects that are funded by IST) is to find, establish and improve synergies of European Grid projects. Members of WP2 have actively participated in the EU deliverables of GRIDSTART as well as working on relations with other EU projects such as EU CrossGrid project.

3.8.5. GLOBAL GRID FORUM

WP2 members have actively contributed to standardisation processes via the Global Grid Forum (GGF) in the fields of data replication and data access. This has also been documented within the GRIDSTART activity. See also deliverable D11.6.

3.8.6. LCG

The LHC Computing Grid project as the main user of EDG has contributed directly to WP2 by providing extra manpower in order to enable us to meet their additional requirements, thus enhancing the overall performance of WP2. This has been a very beneficial, symbiotic collaboration.

3.8.7. PPDG, Griphyn and iVDGL

We have had very close contact with the three U.S. HEP Grid projects PPDG, Griphyn and iVDGL from the beginning of the EDG project. They have contributed to the early requirement and assessment of the EDG testbed. Later they coordinated their efforts with WP2 through LCG.

3.9. Work Package Management and Internal Project Activities

WP2 has six actively contributing partners: CERN, INFN, PPARC, SRC, IRST, UH. The lead partner was CERN, providing the largest amount of manpower and from where the workpackage was run. In the following we will discuss issues arising during the lifetime of the project and how they have been dealt with in WP2. We will not go into the financial details of WP2, and refer the reader to the quarterly and yearly reports.

3.9.1. Personnel

In a project of limited duration it is very common that people leave early or join the project half-way through. During the lifetime of the project we had a total of 50 people working in WP2, the active people being between 20 and 30 (many of them not more than 50%). This means that we had a lot of migration and had to spend considerable time in bringing newcomers up to date.

Hence it is only natural that we were one of the initiators of the EDG tutorial program and that the very first EDG tutorial was given by the WP2 workpackage manager. By streamlining the learning process of new members we could keep the overhead lower and actually achieve our tasks in the given timeframe.

3.9.2. Communication

Working in a distributed environment has the obvious challenge that people get out of touch or get behind in following the activities of the project. We made the usage of mailinglists, phone conferences and regular workshops part of our daily routine in order to address these issues. We have found it essential to hold regular face-to-face workshops in order keep everyone up-to-date and synchronized. There is a limit on how much you can do using phone conferences and over email.

3.9.3. Project reorientation towards production

The project has decided to focus on production issues midway through its lifetime. This was a serious challenge for WP2 as some of its unfunded members are students willing to work on prototyping but not necessarily on service support. Nevertheless, the middleware was produced and delivered with the necessary QoS.

4. EDG DATA MANAGEMENT ARCHITECTURE

The data management architecture, covering all the components and services developed for, deployed and integrated into the final EU DataGrid release, is presented here.

Throughout the EDG project WP2 followed a software development process that involved prototyping, testing, deployment into the application testbed and then redesign and improvement in subsequent releases based on user feedback.

The first generation code was mainly written in C++ whereas the second generation, and final, code was mainly written in Java, adhering to the following principles [12]:

- Modularity: Data management components were designed to be modular to easily allow for plug-ins and future extensions. In addition, we endeavoured to adhere to agreed standards and not rely upon vendor specific solutions,
- Evolution: Since the upcoming OGSA (Open Grid Service Architecture [33]) standard is most likely to be adopted by Grid services in the future, the design should allow for an easy adoption of the OGSA concept.
- Deployment: A vendor neutral approach was adopted for all components to allow for different deployment scenarios. The data management services are independent of the underlying operating system and have been tested on Tomcat and Oracle application servers, interfacing to both MySQL and Oracle database back-ends.

The technology choices for the second generation services presented in this document are as follows:

- Services are implemented in Java and hosted in a J2EE (Java 2 Platform, Enterprise Edition) application server, either Tomcat4 or Oracle 9i Application Server.
- Interface definitions and APIs are exposed via WSDL (Web Service Description Language [30]),
- Java and C/C++ APIs are exposed to clients via through SOAP using Axis and gSOAP, respectively. Command Line Interfaces are also provided.
- Persistent data is stored in a relational database management system. Services that make data persistent have been tested and deployed with both open source, MySQL, and commercial, Oracle 9i, database back-ends. [1]

The EDG Data Management architecture comprises the following logical collections of services:

- **Replication Services**: These services form the core part of the data management services. The replica management system includes the following services: Replica Location Service, Replica Metadata Catalog, and the Replica Optimization Service. The primary user-facing interface to these services is the Replica Manager client.
- **Database Access Service**: The Spitfire tool provides a means to access relational SQL databases from the Grid.
- **Security Layer**: All data management services have very strict security requirements. The Java Security Package provides tools that can be used with Grid services such as our replication services.

These services are discussed in detail in the following sections.

[1] The LCG project has chosen to deploy the WP2 catalog services using the Oracle 9i application server and Oracle 9i database.

4.1. Web Service Design

Web service technologies [23, 24] provide an easy and standardized way to logically connect distributed services via XML (eXtensible Markup Language) messaging. They provide a platform and language independent way of accessing the information held by the service and, as such, are highly suited to multi-language, multi-domain environments such as DataGrid. The emerging OGSA standard [33] aims at leveraging web services in the Grid context for building an open, flexible and extensible grid infrastructure.

All the data management services have been designed and deployed as web services and are implemented in Java. The services run on Apache Axis[27] inside a Java servlet engine. All services use the Java reference servlet engine, Tomcat[28], from the Apache Jakarta project[29]. The Replica Metadata Catalog and Replica Location Service have also been successfully deployed into the Oracle 9i Application Server and are being used in production mode in the LCG project.

Some of the WP2 services make use of a RDBMS (Relational Database Management System) for the storage of persistent data. The installation of the RDBMS is independent of the web service, allowing an existing RDBMS to be used if so desired. The local database administrator retains full control of the database back-end, with only limited administration rights being exposed to properly authorized Grid users. The connection of web services to databases is done through the Java Naming and Directory Interface (JNDI)[31]. We have deployed the WP2 services using MySQL (in EDG) and Oracle (for LCG). IBM has started to invest effort into testing the DB2 back-end in collaboration with WP2.

All services expose a standard interface in WSDL format[30] from which client stubs can be generated automatically in any of the common programming languages. A user application can then invoke the remote service directly. Pre-built client stubs are packaged as Java JAR files and shared and static libraries for Java and C/C++, respectively. The C/C++ clients provide significant performance benefits and are built based on the gSOAP toolkit. The WP2 security task has extended the functionality of the server and clients to add a security layer accepting and managing Grid certificates as described in Section 4.4..

The communication between the client and server components is via the HTTP(S) protocol. This maximises the portability of the service since this protocol has many pre-existing applications that have been heavily tested and are now very robust. The data format is XML, with the request being wrapped using standard SOAP Remote Procedure Call (RPC).

4.2. Replication Services

In the this section we first give an architectural overview of the replication framework and then discuss each individual service in more detail.

Figure 1 presents the user's perspective of the main components of the replica management system. This design, which was first presented in [17, 18], represents an evolution of the original design presented in [3, 7]. Several of the components have already been implemented and tested in EDG, whereas others (shaded in the figure) formed part of the original design and might be implemented in the future by follow-on projects to EDG.

The design of replica management system is modular allowing for easy plugability of third party components, thus defining the minimal interface that third party components must provide. Users and other services access the replica management services via the **Replica Manager**, a logical single point of entry to the system.

The **Replica Manager** is implemented as a client side tool. The Replica Metadata Catalog, Replica Location Service and the Replica Optimization Service are all stand-alone services in their own right, allowing for a multitude of deployment scenarios in a distributed environment.

One advantage of such a design is that if any service is unavailable, the Replica Manager can still provide the functionality that does not make use of that particular service. Also, critical service components may have more than one instance to provide a higher level of availability and to avoid service bottlenecks.

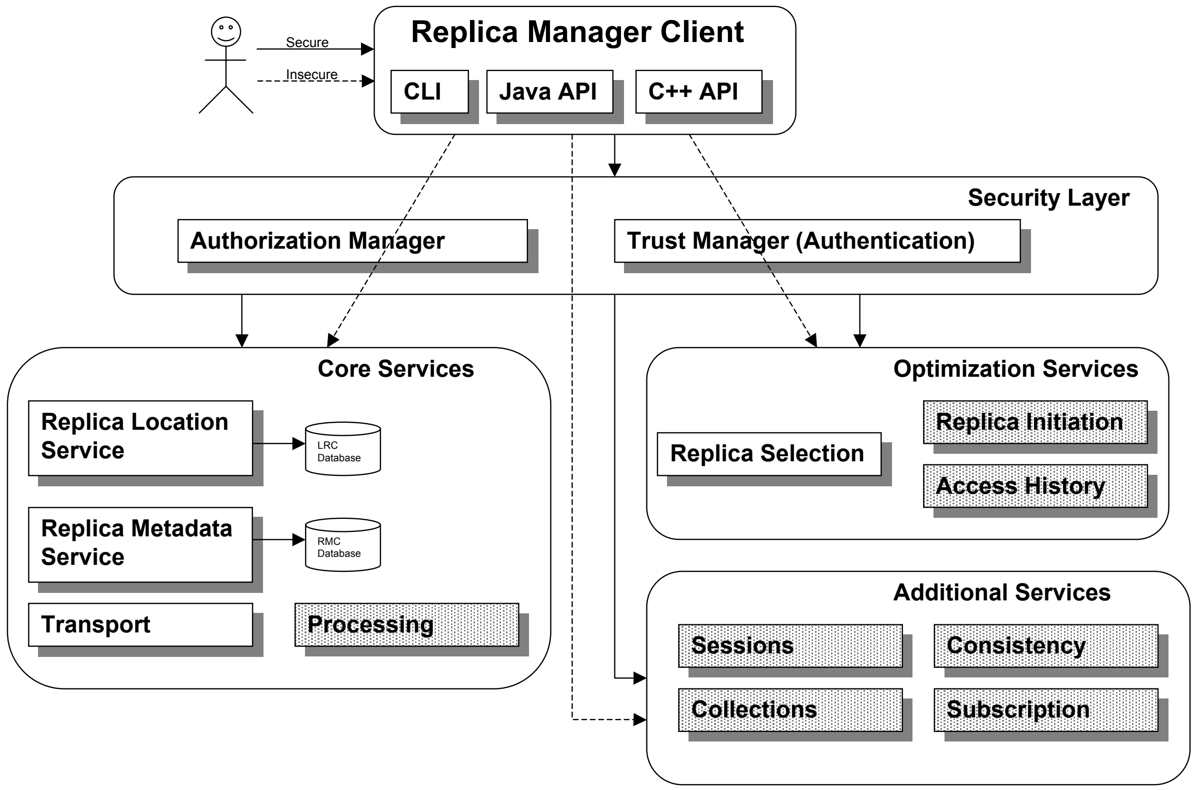

Figure 1: Components of the replica management system. Shaded components have not been implemented in EDG/WP2.

The **Replica Manager** coordinates the interactions between all components of the systems.

- The **Core** functionality of the replica management system of replica creation, deletion, querying, and cataloguing, is provided by the Replica metadata Catalog and Replica Location Service together with the transport services.
- Optimizing access to replicas is provided by the **Optimization** components. The purpose of the optimization service is to minimize file access times by directing access requests to appropriate replicas. Pro-active replication of frequently used files, based on gathered access statistics, could also prove to be useful once it is based on data access patterns from the application areas.
- The **Security** module manages the required user authentication and authorization, in particular, issues pertaining to whether a user is allowed to create, delete, read, and write a file.
- **Collections** are defined as sets of logical filenames and other collections.
- The **Consistency** module maintains consistency between all replicas of a given file, as well as between the meta information stored in the various catalogs.

- The **Session** component provides generic check-pointing, restart, and rollback mechanisms to add fault tolerance to the system.
- The **Subscription** service allows for a publish-subscribe model for replica creation.

A detailed description of the implemented components and services can be found in the following sections and in the original design [17].

4.2.1. Replica Manager

The Replica Manager makes use of many services to carry out its tasks. The replica management system requires both services internal to data management, discussed here, and external services. Examples of the necessary external services are an Information Service such as MDS (Monitoring and Discovery Service) or R-GMA (Relational Grid Monitoring Architecture), storage resources such as SRM (Storage Resource Manager) or the EDG-SE (EDG Storage Element), and transport mechanisms such as GridFTP. Most of the components required by the Replica Manager are deployed as independent services, hence the appropriate client stubs compliant to the agreed upon interfaces need to be provided by the service.

By means of configuration files the actual component to be used can be specified and Java dynamic class loading features are exploited to make them available at execution time.

The Replica Manager deployed in the final release of the EDG software makes use of the following services:

- *Replica Location Service (RLS)* [19]: used for locating replicas in the Grid and assigning physical file names.
- *Replica Metadata Catalog (RMC)*: used for querying and assigning logical file names.
- *Replica Optimization Service (ROS)*: used for locating the best replica to access.
- *R-GMA*: an information service provided by EDG: The Replica Manager uses R-GMA to obtain information about Storage and Computing Elements [18].
- *MDS*: Globus' information service based on LDAP. Note that either MDS *or* R-GMA is used.
- *Globus C based libraries via JNI or CoG* [10] providing GridFTP transport functionality.
- The *EDG network monitoring services*: EDG (in particular WP7) provides these services to obtain statistics and network characteristics.
- *Storage Services*; the Replica Manager interacts with both the EDG Storage Element and the classic "GridFTP" servers for storing data.

The implementation is mainly done using the Java J2EE framework and associated web service technologies (the Apache Tomcat servlet container, Jakarta Axis , etc.). In more detail, client/server architectures making SOAP Remote Procedure Call (RPC) over HTTP(S) are used. The basic component interaction is given in Figure 2.

For the user, the main entry point to the Replication Services is through the client interface that is provided via a Java API and a command line interface. For each of the main components in Figure **??**, the replica management system uses the necessary interface.

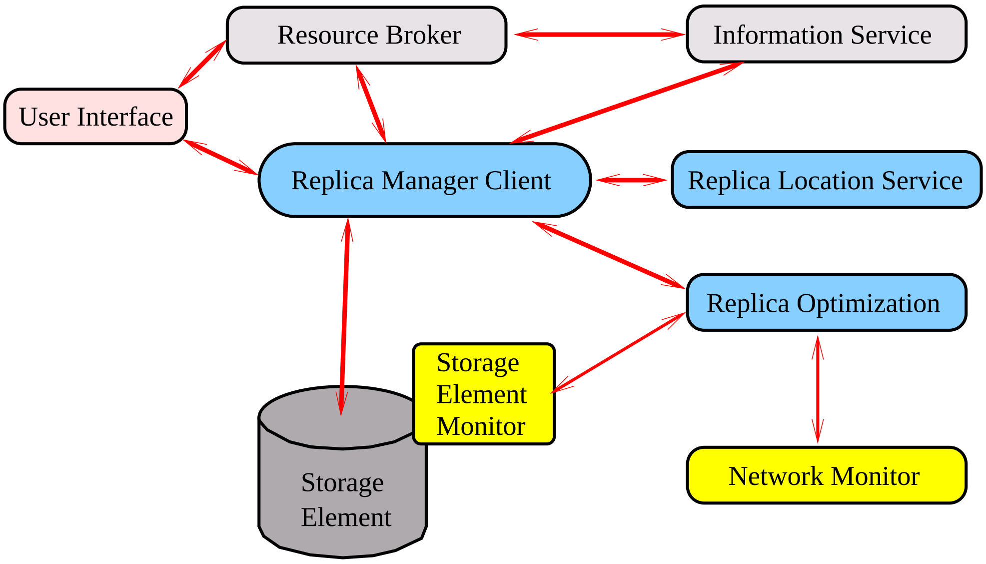

Figure 2: Interaction of Replica Manager with other Grid components.

4.2.2. Replica Location Service (RLS)

In a highly geographically distributed environment, providing global access to data can be facilitated via replication, the creation of remote read-only files. In addition, data replication can reduce access latencies and improve system robustness and scalability. However, the existence of multiple replicas of files in a system introduces additional issues. The replicas must be kept consistent, they must be locatable and their lifetime must be managed. The Replica Location Service (RLS) is a system that maintains and provides access to information about the physical locations of copies of files [19].

The RLS architecture defines two types of components: the Local Replica Catalogs (LRC) and the Replica Location Indices (RLI) (see Figure 3). The LRC maintains information about replicas at a single site or on a single storage resource, thus maintaining reliable, up to date information about the independent local state. The RLI is a (distributed) index that maintains soft collective state information obtained from any number of LRCs.

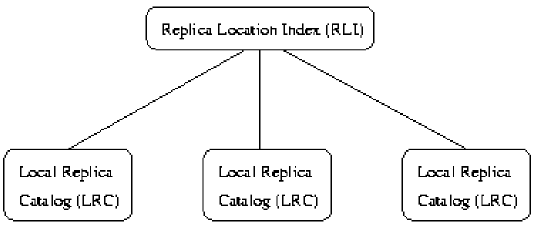

Figure 3: Simple RLS structure

The advantages of having such an RLS structure are:

- The presence of local LRCs enable the quick lookup of locally available files. Most of the files that are necessary for a typical job will be pre-fetched so we expect the local lookup situation to be the most frequent one in the HEP environment.
- Having distributed RLIs allows for a greater flexibility, as they can be set up on demand. By distributing the load of lookups this way, we can enhance the scalability of the system.

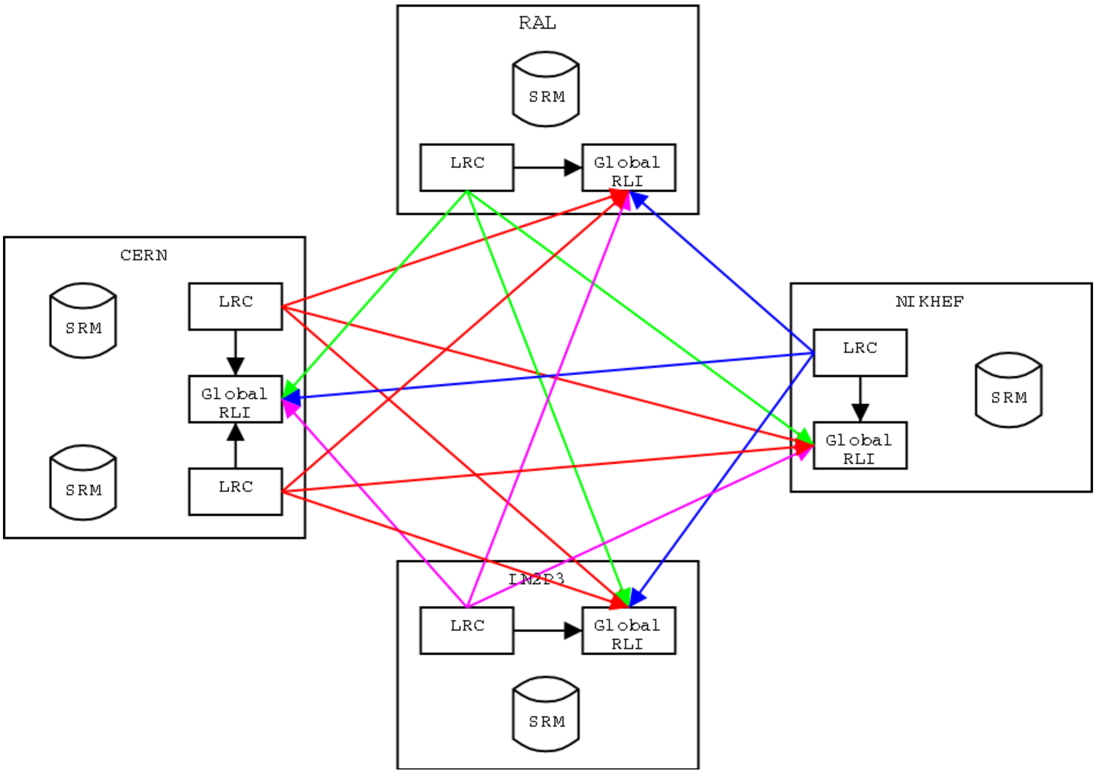

Figure 4: A possible RLS structure with four sites, each with at least one LRC and one global RLI.

The LRC stores the mappings between GUIDs and SURLs, whereas the RLI stores mappings between GUIDs and LRCs, thus identifying all the LRCs that store a physical replica of a given GUID. A query on a replica is a two stage process. The client first queries the RLI in order to determine which LRCs contain mappings for a given GUID. One or more of the identified LRCs is then queried to find the associated SURLs.

An LRC is configured at deployment time to subscribe to one or more RLIs. The LRCs periodically publish the list of GUIDs they maintain to the set of RLIs that index them using a soft state protocol, meaning that the information in the RLI will time out and must be refreshed periodically. The soft state information is sent to the RLIs in a compressed format using bloom filter objects.

An LRC is typically deployed on a per site basis, or on a per storage resource basis, depending on the site's resources, needs and configuration. A site will typically deploy 1 or more RLIs depending on usage patterns and need. A possible RLS deployment scenario is shown in Figure 4.

The LRC can be deployed to work in stand-alone mode instead of in fully distributed mode, hence providing the functionality of an replica catalog operating in a fully centralized manner. In stand-alone mode, a VO would deploy one central LRC that held the GUID to SURL mappings for all the VO's distributed grid files.

The LRC and RLI implementations support both MySQL and Oracle9 database back-ends for persistent storage of mappings and can be hosted within either the Tomcat 4 or the Oracle 9i Application server environments.

4.2.3. Replica Metadata Catalog Service (RMC)

The GUIDs stored in the RLS are neither intuitive nor user friendly. The Replica Metadata Catalog (RMC) allows the user to define and store LFN aliases to GUIDs. Many LFNs may exist for one GUID.

In addition, the RMC can store GUID metadata such as file size, owner, ACL and creation date. The LFN needs to be unique within the RMC.

The RMC is not intended to manage all generic experimental metadata however it is possible to extend the RMC to maintain some user definable metadata to around O(10) items per Virtual Organization. This metadata provides a means for user to query the file catalog based upon application-defined attributes.

The possibility of managing LFNs as collections and manipulating these collections as a whole is discussed in the future work in section 7.

The RMC implementation uses the same technology choices as the RLS, and thus supports different back-end database implementations, and can be hosted within different application server environments.

Figure 5 shows how the mappings are stored in the RLS and RMC catalogs.

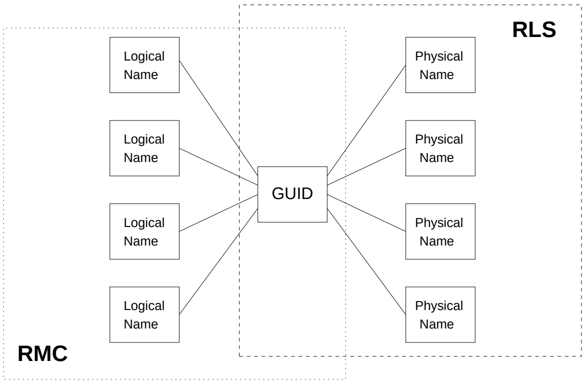

Figure 5: The Logical File Name to GUID mapping is maintained in the Replica Metadata Catalog, the GUID to Physical File Name mapping in the RLS.

4.2.4. Replica Optimization Service (ROS)

Optimisation of the use of the grid resources such as computing elements, network resources, storage resources and data management resources is essential for application jobs to be executed efficiently.

The Replica Optimization Service (ROS) focuses on the selection of the best replica of a data file for a given job, taking into account the location of the computing resources and network and storage access latencies.

The Network monitor provides the `getNetworkCosts` API [20] that is used by the Replica Optimization Service to obtain information on network latencies between the various grid resources. Similarly, the Storage Element Monitor provides the `getSECosts` API that ROS uses to obtain information about file access latencies at a given site. Together this information is used to calculate the expected transfer time of a given file with a specific size.

In EDG, Grid resources are managed by the meta scheduler of WP1, the Resource Broker [6] as part of the Workload Management System. One of the goals of the Workload Management System is to decide on which Computing Element a job should run such that the throughput of all jobs is maximized. Assuming highly data intensive jobs, a typical optimization strategy could be to select the least loaded resource with the maximum amount of locally available data. In [20] we introduced the Replica Manager API `getAccessCost` that returns the access costs of a specific job for each candidate Computing Element. The Workload Management System can take into account this data location information provided by the Replica Manager when make its scheduling decision.

The ROS is implemented as a light-weight web service that gathers information from the EDG network monitoring service. Collaboration with the CrossGrid project has led to the integration of their service for estimating storage access costs.

The interaction of the Replica Manager with the Resource Broker, the Network Monitor and the Storage Element Monitor is depicted in Figure 2.

4.3. DATABASE ACCESS SERVICE, SPITFIRE

Spitfire [25] is an SQL Database service that provides a means to access relational databases from the Grid. Spitfire permits convenient and secure storage, retrieval and querying of data held in any local or remote RDBMS. The service has been optimized for metadata storage and queries.

The design of the service is similar to that of the prototype RPC GridDataService [21], and indeed influenced the design of this standard. It is expected that the SQL Database service will eventually evolve into a implementation of parts of this GGF standard. The architecture of the service is such that it will be easy to implement the OGSA specification at a later date.

There are three main components to the SQL Database service: the primary server component, the client component, and the browser component. Applications link to the SQL Database client library to communicate with a remote instance of the server; the API is designed around the SQL query language. This server is deployed in front of a RDBMS, such as MySQL, and securely mediates all Grid access to that database. The browser is a stand-alone web portal that is also deployed in front of a RDBMS. The communication between the user's web browser and the SQL Database Browser service is over HTTP(S).

The original SQL Database service was primarily accessed via a web browser or command line interface using pre-defined server-side templates. This functionality was found to be very useful for web portals, providing a standardized view of the data. It has been retained and re-factored into a separate stand alone SQL Database browser module, usable from any web browser. It is implemented as a Java servlet.

The core SQL Database service was re-designed as a web service. The interface provides the common SQL operations to work with the data. Pre-built client stubs exist for the Java, C and C++ programming languages. The service itself has been tested with the MySQL and Oracle databases.

Both, the server and browser have a service certificate (optionally they can make use of the system's host certificate), signed by an appropriate Certificate Authority, which is used to authenticate to the client. Clients use their GSI proxy to authenticate to the service. To use the SQL Database browser service, users must load their GSI certificate into their web client, which will then be used to authenticate the user to the SQL Database browser.

In the case where the Database browser and core Spitfire service are installed on the same machine, it is envisaged that both services will be installed inside the same servlet engine.

A basic authorisation scheme is defined by default for the SQL Database service, providing administrative and standard user functionality. The authorisation is performed using the subject name of the user's certificate (or a regular expression matching it). The service administrator can define a more complex authorisation scheme if desired, as described in the security module documentation.

4.4. SECURITY PACKAGE

The analysis of the security requirements for data management middleware services identified two main areas of concern:

- Secure access to web services through the use of reliable authentication and authorization;
- Improvement of security of the CASTOR [9] mass storage system through the use of reliable authentication.

The Java Security Package was developed to address secure access to web services while a solution based on GSI authentication was developed together with the CASTOR team for the CASTOR system.

4.4.1. Java Security Package

The aim of the edg-java-security package is to provide a robust security infrastructure for Java-based web services while not compromising on software flexibility. It completely fulfils all the requirements identified for the provision of secure access to WP2 services.

The original design proposed three complementary solutions to achieve secure access to the data management services:

1. HTTP over SSL as the communication protocol between clients and servers. SSL provides mutual authentication of clients and servers.
2. Tomcat security realms as a mechanism to control access to Web application resources.
3. Extension of the XSQL access control mechanism to use information included in X.509 certificates for the authorization decisions and to dynamically embed authorized attributes in the XSQL request.

The final release follows a rather different approach to that of the original design. The use of Tomcat security realms and the XSQL access control mechanism was replaced by the wider concept of securing any web service hosted in Tomcat. Security realms are a solution not scalable and manageable on the long term and the XSQL access control mechanism is focused only on the protection of database applications.

The wider approach implemented by WP2 addresses the two security areas of authentication and authorization. Authentication ensures that the entity (user, service or server) at the other end of the connection is who it claims to be. Secure authentication is always a two-way process. Both sides (client and server) have to be authenticated to one another. The server needs to authenticate all clients but the client also needs to know whether it talks to the right service. Authorization decides what an entity is allowed to do. The EDG Java security package consists of two main components: one completely dedicated to Authentication of any service requests (TrustManager) and one completely dedicated to Authorization of any service requests (AuthorizationManager). The result is a set of extremely modular and configurable security mechanisms that can reliably protect any web service hosted in the Tomcat servlet engine.

Emerging industry standards have been taken into account where applicable to make the software usable everywhere. To this end, there has been some research into similarities and possibilities for cooperation with, for example, the Liberty Alliance (`www.projectliberty.org`), which is a consortium developing standards and solutions for federated identity for web based authentication, authorization and payment. WP2 have been active in GGF security activities and so contribute to the formation of grid standards in the area of security

Authentication The authentication mechanism developed by EDG is an extension of the standard Java implementation of the SSL/TLS protocol. Mutual authentication in SSL is based on public-key certificates that are signed by trusted Certification Authorities (CA). The user and the server verify each others' identities by proving in cryptographic means that they have access to the private key that matches with the certificate.

The Globus Toolkit introduces the use of GSI proxy certificates. A proxy certificate consists of a new certificate signed by the end entity (user or server) that created it, rather than by a CA. This proxy certificate comes close to fulfilling the *PKIX* [22] requirements for a valid certificate chain, but does not fully adhere to the standard since it is actually signed by an entity that is not a CA. This causes the mutual authentication to fail in any implementation that fully conforms to the SSL standard. For the GSI proxy to work, the SSL implementation has to be changed to a non-standard form that accepts proxy certificates.

The EDG Java security package extends the Java SSL package. It

- accepts and verifies GSI proxy certificates
- supports GSI proxy loading with periodical reloading
- supports OpenSSL certificate-private key pair loading
- supports CRLs with periodical reloading
- is integrated with Tomcat for both HTTP and SOAP communication
- is integrated with Jakarta Axis SOAP framework for SOAP communication
- is integrated with HTTPClient for HTTP communication
- can be used to create normal sockets for plain TCP communication

GSI proxy support is done by parsing the certificate chain in order to find the end entity certificate and enforcing special allowances and restrictions to the following (proxy) certificates in the chain. The allowance is that a proxy certificate does not have to be signed by a CA. The restriction is that the distinguished name (DN) of a proxy certificate has to start exactly with the DN of the end entity that signed it. E.g. the entity 'C=CH, O=cern, CN=John Doe' can generate a proxy with DN 'C=CH, O=cern, CN=John Doe, CN=proxy' but cannot pretend to be someone else by generating a proxy with DN 'C=CH, O=cern, CN=Nicole Doe, CN=proxy'.

Proxy certificates are short lived, usually 12 to 24 hours, so a program using them may still be running while the proxy is updated. For this reason the credentials (the proxy certificate and its associated private key) can be periodically reloaded in a fully automatic way.

OpenSSL saves the end entity credentials in two files, one for the certificate and one for the private key. The EDG Java security package provides functions to easily load these credentials.

Certification Authorities periodically issue lists of revoked certificates (CRL). The EDG Java security package supports verification of certificates against these CRLs and can also periodically and automatically reload the CRLs by simply setting a reload interval.

The integration with Jakarta Tomcat is done providing a specific implementation of the ServerSocketFactory interface. The use of the GSI-enabled socket implementation is activated by setting up accordingly the Jakarta Tomcat configuration file.

The integration with the Jakarta Axis SOAP framework was even simpler, since the Axis server runs on top of Tomcat and Tomcat can be set up as above. On the client side the Axis framework provides an easy way to change the underlying SSL socket implementation: when calling the Java program one system property has to be set to indicate which secure socket factory must be used.

C++ SOAP clients for data management services are built based on the gSOAP toolkit, so in order to provide the same authentication and authorization functionality as in the corresponding Java SOAP clients, an accompanying C library was developed for gSOAP. This library provides support for mutual authentication between servers and clients, verification of both standard X.509 and GSI proxy certificates and support for coarse-grained authorization as implemented in the server end by the Authorization Manager.

Coarse grained authorization The EDG Java security package implements a coarse-grained authorization model. The authorization mechanism is positioned in the server-end in front of the service and the authorization decision is made before the actual call to the service. Coarse-grained authorization means the system can make decisions such as: 'what kind of access does the user have on a specific database table?' or 'what kind of access does the user have on the file system?'. Fine-grained authorization, which can make decisions such as: 'what kind of access does the user have on this file?', can only be handled inside the service because the actual file to access is only known during the execution of the service.

The EDG Java security package implements a role-based access control policy. The authorization is performed in two stages:

1. The system checks that the user is permitted the role requested. If the user does not request any role, the system checks for any default defined role.
2. The role that the user is authorized to use is mapped to a service specific attribute.

The role definitions can be the same for all the services in the virtual organization, but the mapping from role to attribute is service specific. The service specific attribute can be, for example, a user-id for a file system access or a database connection-id with pre-configured access rights. If either of the above two steps fail, the user is not authorized to access the service using the requested role.

The authorization system consists of

- an authorization engine where access control policies are stored and authorization decisions are taken;
- two wrappers, one for HTTP messages and another for SOAP messages, that interface the engine to the communication flow between the client and the service and enforce the authorization decisions.

The authorization engine can be attached to other communication flows by writing the appropriate interface wrapper for them.

In the authorization engine the information that is used to make the authorization decisions can be obtained from several different sources. For simple and small installations and for testing purposes, the information can be stored in a XML file. For larger installations the information can be stored in a database. If the Globus tools are used to distribute the authorization information, data is normally stored in a text file known as the grid-mapfile. If the Virtual Organization Membership Service (VOMS) is used, the information included by the VOMS server in the proxy certificate can be used for the authorization decisions. For each of these stores there is a module to handle the specifics of that store; to add a new way to store the authorization information only the appropriate interface module needs to be written.

The authorization wrappers parse any incoming service requests to extract the DN of the client and requested role, if requested. The DN and role are forwarded to the authorization engine where the access control decision is taken. According to the result of this operation the wrappers either deliver the request, which now contains the service specific attribute, to the destination service or block and then terminates the request.

Administration web interface The authorization information usually ends up being rather complex, and might become rather difficult to be maintained manually. For this purpose a web based administration interface was is included in the EDG Java security package. This tool is helpful to examine the authorization configuration and enables remote management of the authorization system. Moreover, making management easier improves the security.

A specific access control policy is enforced by the web interface in order to guarantee that only appointed administrators are allowed to view/edit the system configuration.

4.4.2. CASTOR SECURITY

WP2 proposed different solutions to provide secure access to CASTOR:

1. Abandon the use of userID/groupID, sent in clear-text, as the mechanism to verify client identity and access rights.
2. Introduction of GSI mutual authentication between clients and front-end servers.
3. Introduce of GSI mutual authentication between front-end servers and back-end servers.

The integration of a GSI-enabled authentication mechanism into CASTOR was started by WP2. The code has been handed over to the CASTOR team who have finalized this work to render CASTOR into a GSI-enabled service.

5. Evaluation of Data Management Services and Components

In the following sections each of the services and components of the final release is evaluated in detail. The evaluation is done using the following criteria:

- **Functionality/Integration**: This item covers whether the required functionality has been met and how the integration is done with other components.
- **Performance**: For all clients or client side tools the wall clock time for execution of various operations within the tool as seen by the end user is measured.
- **Scalability**: Scalability evaluation was done for the services in order to measure the maximum amount of parallel requests, entries in catalogs, etc.

At this point let it be mentioned that the command line interfaces that have been tested for performance are all written in Java, i.e. they take a considerable performance hit when starting up the JVM. The reason why Java was chosen to implement the CLI and not C++ was that we were delayed in providing the C++ API to our services due to the complexity of the security interaction through an autogenerated SOAP client. Because of this delay we had to provide the CLI through our Java API. By rewriting the CLI using the C++ API a considerable performance increase may be achieved, unfortunately we ran out of time in EDG to provide it. We have been focusing on functionality and stability in the last year, not on performance.

5.1. Testbed Setup

For our performance tests we used the WP2 testbed consisting of 13 machines and 5 sites. All the machines are Intel Pentium PCs running GNU/Linux RedHat 7.3.3 (or 7.3.2). The list of machines is given in Table 1.

Machine	Location	Clock Speed	Memory	Function
lxshare0313	CERN, Switzerland	Dual 1 GHz	512 MB	Client, LRC, ROS
lxshare0344	CERN, Switzerland	Dual 1 GHz	512 MB	Client
lxshare0343	CERN, Switzerland	Dual 1 GHz	512 MB	LRC, RMC
lxshare0346	CERN, Switzerland	Dual 1 GHz	512 MB	LRC, RMC
pccms144	CERN, Switzerland	200 MHz	96 MB	LRC
pcrd24	CERN, Switzerland	Dual 600MHz	512 MB	Storage Estimator
zeus04	Krakow, Poland	Dual 1GHz	512 MB	Storage Estimator
grid04	Glasgow, Scotland	1.25 GHz	512 MB	LRC, RLI
grid05	Glasgow, Scotland	1.66 GHz	512 MB	LRC, RLI
grid06	Glasgow, Scotland	1.66 GHz	512 MB	LRC, RLI
is01vidgrid	Vienna, Austria	200 MHz	64 MB	LRC
grid	Innsbruck, Austria	1.66 GHz	512 MB	LRC
eio01	Tel Aviv, Israel	450 MHz	256 MB	LRC

Table 1: Testbed used for evaluation.

For most of the tests we used Java 2 or C++ (gcc 3.2.2). On the server side, the services LRC (v2.1.2), RLI (v2.1.3), RMC (v2.1.2) and ROS (v2.1.4-cg) run on Tomcat 4.1.18. For storing data on the server side, MySQL 4.0.13 was used.

5.2. Replica Location Service

In the following sub-section we provide a set of evaluations for the Replica Location Service. The single Local Replica Catalog and full Replica Location Service with multiple Local Replica Catalogs and Replica Location Indexes will be covered separately. All our tests were done *without* using GSI security. The reason is that security adds a considerable performance hit which we would like to measure separately. At this point we are interested in the performance of the actual functionality of the service and how it scales, not in how much authentication and authorization takes, which is a constant. A comparison of performance with and without security can be seen in section 5.6.

5.2.1. Functionality/Integration

Within the EDG testbed, the RLS has so far been used with a single LRC per VO. This has the advantage that query results are highly performant but it also creates a single point of failure. However, we have performed tests with a single LRC as well as a complete LRC/RLI system.

One of the open issues is to make the RLS interoperable with the RLS provided by the Globus Toolkit™.

5.2.2. Performance - Local Replica Catalog Only

In the following section we test the insert, query and delete rates of the Local Replica Catalog as seen by the end user. For all our tests we used the WP2 testbed (Table 1) with clients and servers on the same local area network. For most of the performance tests we developed small test applications; these are packaged with the software and can therefore be re-run to check the results obtained. Results using C++, Java and command line clients are shown in the following sections.

C++ API

To test the performance using a C++ client, the client and server machines used were lxshare0313 and lxshare0343 respectively. The performance test suite first inserts from 10 to 10,000,000 GUID:PFN mappings, checks for existence, queries an entry and then deletes them. This tests how each of these operations on the LRC scales with the number of entries in the catalog.

Figure 6(a) shows the total time to insert and delete a certain number of mappings, and Figure 6(b) shows how the time to query one entry varies with the number of entries in the LRC.

The results show that insert and delete operations show very stable behaviour, in that the total time to insert or delete mappings scales linearly with the number of mappings inserted or deleted. A single transaction with a single client thread takes about 25 - 29 ms with the tendency that delete operations are slightly slower than inserts. Queries show a constant behaviour i.e. the query time is independent of the number of entries in the catalog up to 1 million entries, when it tends to increase.

Java API

To simulate many users concurrently using the LRC, a Java client class which enabled multiple threaded operations was used. A defined number of threads from a thread pool performed the specified number of insert, query and delete operations as quickly as possible. For each type of operation the average time to execute the operation was measured for every 1000 operations.

Firstly, the time to insert a GUID:PFN mapping into the LRC as a function of number of entries in the LRC and number of concurrent threads was measured. Figure 7 shows, for different numbers of concurrent threads, how the average time to insert a mapping varies over the course of inserting 500,000 map-

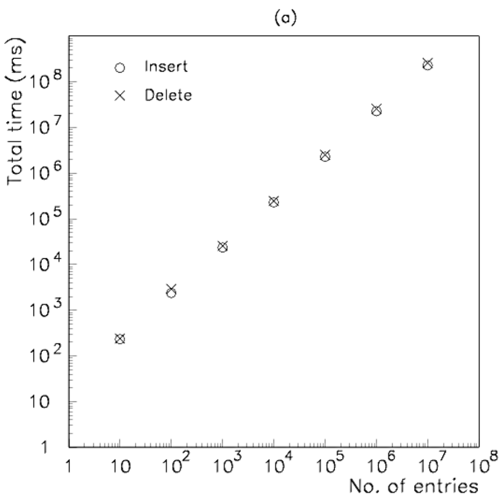

Figure 6: (a) Total time to add and delete mappings and (b) query the LRC using the C++ API.

pings into an empty LRC. The machines used for these measurements were lxshare0344 and lxshare0346 at CERN, and grid04 and grid06 at Glasgow.

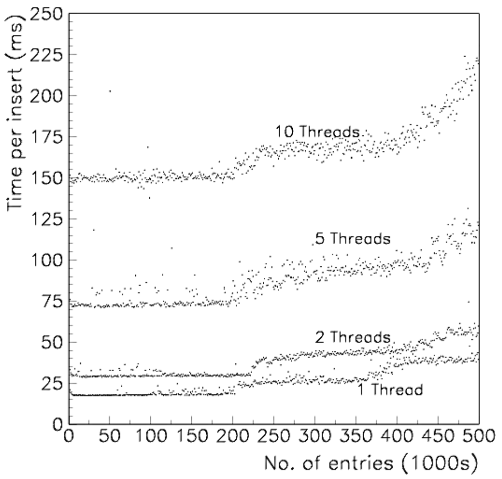

Figure 7: Time to insert one mapping into the LRC as the number of entries and number of concurrent threads varies.

The shapes of the plots are very similar for the different number of concurrent threads, in that the insert time is constant for about the first 200,000 entries, but after this it gradually increases as the number of entries increases. The effect is more pronounced with a lower number of threads, for example with one thread the increase is almost 100% between 0 and 500,000 entries, whereas it is only around 20% when 195 threads are used. However, the data becomes more noisy when more threads are used which tends to obscure the trend.

To measure the effective throughput of the LRC, the total time to insert 500,000 mappings was measured for different numbers of concurrent threads.

Figure 8 shows that the time falls rapidly with increasing numbers of threads, bottoming out after 10 or

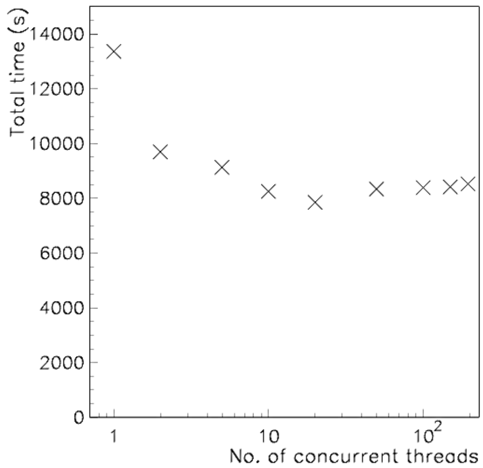

Figure 8: Total time to add 500,000 mappings to the LRC using concurrent threads.

20 threads. For 20 threads the total time taken is about 40% less than using one thread. Although the time for an individual operation is slower the more concurrent operations are taking place, the overall throughput actually increases, showing the ability of the LRC to handle multiply threaded operations.

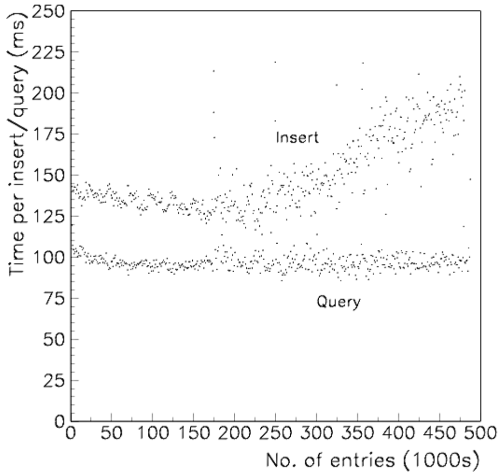

Figure 9: Time to insert mappings and query one GUID for different numbers of entries in the LRC, using 5 concurrent inserting clients and 5 concurrent querying clients.

The variation of the time to query one GUID with number of entries in the catalog was investigated and it was found to be roughly independent of the number of entries (up to 500 thousand) for multiple concurrent clients. Figure 9 compares insert time and query time between a 0 and 500,000 entries LRC. This test was done with 10 concurrent threads, thus at any given moment 5 threads would be inserting a mapping and 5 threads would be querying a mapping. The plot shows the insert time rising from 140 ms to 200 ms as it did in Figure 7 but the query time stays at a constant 100 ms and does not vary with the number of entries.

Some tests were done to simulate realistic situations of users using the LRC in a production environment.

Two typical situations were tested:

- Initially adding data to a catalog: many inserts, a few queries and no deletes, starting from an empty catalog.
- Normal operation (by users doing physics analysis for example): a few inserts, many queries and a few deletes starting from a reasonably full catalog.

Results simulating the first situation are shown in Figure 10(a). Here 500,000 mappings were inserted and 50,000 queries were done using different numbers of concurrent threads. The results are very similar to Figure 8, showing that the extra 50,000 queries have little effect on the total time. Again, the time bottoms out after around 10-20 threads.

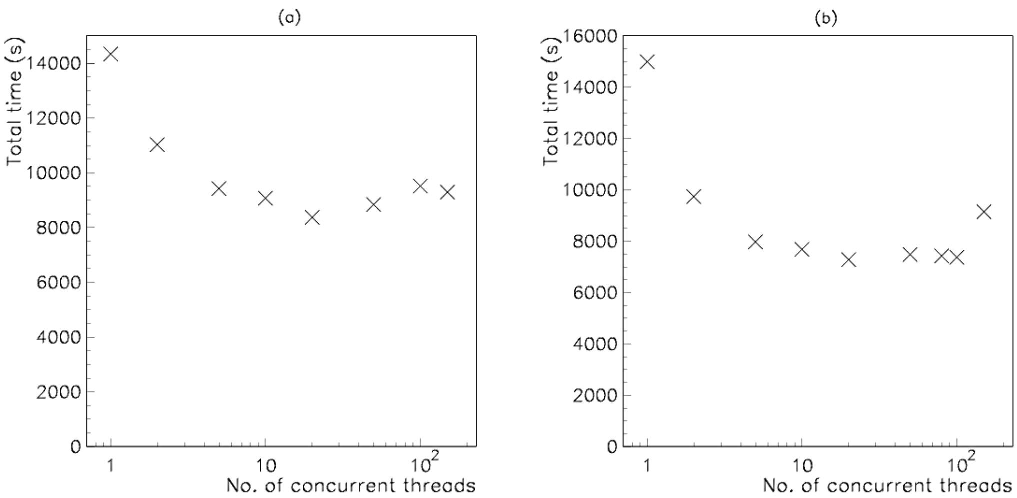

Figure 10: Total time for (a) 500,000 insert and 50,000 query operations, (b) 100,000 insert, 500,000 query and 100,000 delete operations on the LRC using concurrent threads.

Figure 10(b) shows slightly different results. For this plot 100,000 inserts, 500,000 queries and 100,000 deletes were performed on a catalog that initially held 500,000 entries. The minimum total time is not so well defined and even though there were 150,000 more operations than in Figure 10(a) on a large catalog compared to an empty one, the total time is less for all numbers of threads (except 1 thread) than in Figure 10(a). This confirms the result shown in Figure 9, that query operations are much less time consuming than insert operations.

COMMAND LINE INTERFACE

The command line interface is implemented in Java using the Java API. Table 2 shows some timing statistics showing the time to execute different parts of the command addMapping (in insecure mode) used to insert a GUID:PFN mapping into the LRC.

The total time to execute the command was 3.3 seconds and this time is broken down into the following areas: The start-up script sets various options such as logging parameters and the class-path for the Java executable and this along with the time to start the Java Virtual Machine, took 0.8s. After parsing the command line it took 1.3 seconds to get the LRC service locator - during this time many external classes had to be loaded in.

Time (s)	Operation
0 - 0.8	Start-up script and JVM start-up time
0.8 - 1.0	Parse command and options
1.0 - 2.2	Get LRC service locator
2.2 - 2.3	Get LRC object
2.3 - 3.2	1st call to the lrc.addMapping() method
3.2 - 3.3	2nd call to the lrc.addMapping() method
3.3	End

Table 2: Timing statistics for adding a mapping in the LRC using the CLI.

The command was modified slightly so that it would insert two mappings in succession in order to show the effect of the class loading the first time the method is called. It can clearly be seen that this has a large effect: the first call takes around 0.9s but the second time it takes only 0.1s. Thus, for adding multiple entries to the catalog it is far quicker to use multiple calls to the Java or C++ APIs than the command line tool repeatedly, since every time the latter is used the overhead of dynamic class-loading is very large.

In short, the time taken to insert a GUID:PFN mapping using the command line interface is about 2 orders of magnitude longer than the average time taken using the Java or C++ API (see previous sections). Therefore, the command line tool is only recommended for simple testing and not for large scale operations on the catalog. As explained above, this is due to the JVM startup time and our choice to implement the CLI in Java. The C++ and Java performance is in accordance with the requirements, however, the CLI performance needs improving in order to meet the user requirements.

5.2.3. Performance - Full Replica Location Service

The following set of tests use the entire RLS system with at least one RLI and several LRCs.

The first test was done in order to test the query response time of a single RLI using the RLI command line interface. Table 3 shows different tests for retrieving the LRC for a given GUID using the command `getLRCs`. The results show that the query time is constant (about 4 seconds) and independent of the number of LRCs publishing to the RLI.

Number of LRCs	Number of entries in RLI	Response time
2	10,000	4.26s
3	10,000	4.29s
4	10,000	4.26s

Table 3: getLRCs sent to the RLI using the RLI command line interface

The RLI interface does not provide a direct call to query the LRCs; currently only the Replica Manager provides such a functionality. Thus, we report on further RLS tests below using the Replica Manager command line tool.

The following test has been done with the Replica Manager using the full RLS system, i.e. 1 RLI and 1 or many LRCs. We tested queries where replica locations were stored in 1, 2, 3, 4 and 5 LRCs i.e. for 1 to 5 replicas. Since such a query is first sent to the RLI and then an additional redirection step is done to the corresponding LRC that really holds the replica location in the local database, the query time increases slightly with the number of replicas stored in different LRCs. Results are shown in Figure 11.

We have also analysed the latencies and response times of the five LRCs which can vary from time to time since we are using a non-dedicated wide area network link. Table 4 shows that the round trip time

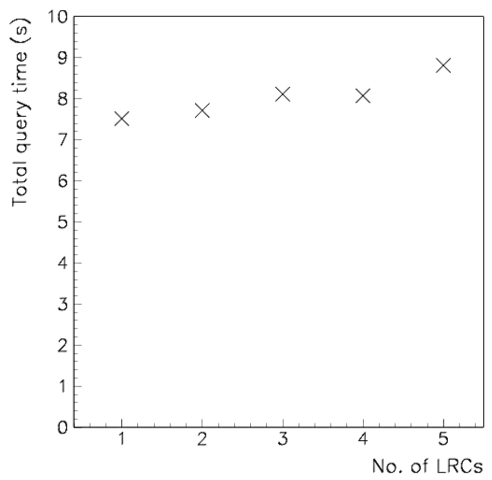

Figure 11: Time to query a RLI for 1 GUID with different numbers of LRCs publishing to the RLI.

(RTT) varies significantly whereas the insert time for a single entry with the command line tool is rather constant due to the overhead imposed by the Java CLI. However, the inserts done with the C++ API show that the 5 severs have different response times, mainly due to the different processors and their speeds. The results show that even machines with lower response time can take part in the RLS system and for single queries in the entire RLS system the main bottleneck is the CLI rather than the server back-end.

Server	RTT	Single insert (CLI)	100 inserts with C++ API
lxshare0313.cern.ch	0.25 ms	4.44 s	19,310 ms
pccms144.cern.ch	0.4 ms	4.45 s	168,370 ms
is01vidgrid.pri.univie.ac.at	20 ms	4.53 s	156,600 ms
grid.uibk.ac.at	27 ms	4.34 s	67,080 ms
eio01.weizmann.ac.il	64 ms	4.67 s	168,260 ms

Table 4: Comparison of the different LRCs used in the RLS system. All queries were executed on the machine lxshare0344.

5.2.4. Scalability

The following questions are covered by the scalability evaluation:

- How many entries can be stored in a single LRC and single RLI?
  Up to 10,000,000 have been successfully tested.
- How many entries can be stored using one RLI and several LRCs?
  This depends on the memory of the RLI (see Chapter 5 "RLI Integration and Tuning" in [40]).
- How many LRCs can be handled by a single RLI?
  Depends on the memory of the RLI (see Chapter 5 "RLI Integration and Tuning" in [40]).
- How many parallel queries can be handled by an RLI/LRC?
  We have successfully tested more than 200 parallel threads.

5.3. Replica Metadata Catalog Service

In the following sub-section we provide a set of evaluations for the Replica Metadata Catalog Service.

5.3.1. Functionality/Integration

The Replica Metadata Catalog can be regarded as an add-on to the RLS system and is used by the Replica Manager to provide a complete view on LFN:GUID:SURL (Figure 5) mapping.

Currently, the service is a single point of failure since it keeps all the LFNs in the one place. The service database can be replicated to increase fault tolerance.

5.3.2. Performance

In the following section we test the insert, query and delete rates of the RMC as seen by the end user. As with the RLS, for all our tests we used the WP2 testbed with clients and servers on the same local area network. For most of the performance tests we developed small test applications; these are packaged with the software and can therefore be re-run to check the results obtained.

C++ API

A C++ (gcc 3.2.2) test program which inserts up to 1,000,000 GUID-LFN pairs into the RMC was used to test the performance of the RMC with various numbers of entries. In the first set of tests, shown in Figure 12, we used a single LFN per GUID and saw very similar insert/delete and query rates as for a single LRC, which is not surprising since both servers are implemented in a very similar way. On average, a single insert operation takes about 22 ms as compared to 24ms of a delete operation using a single client thread. Query operations range between 10 and 20 ms.

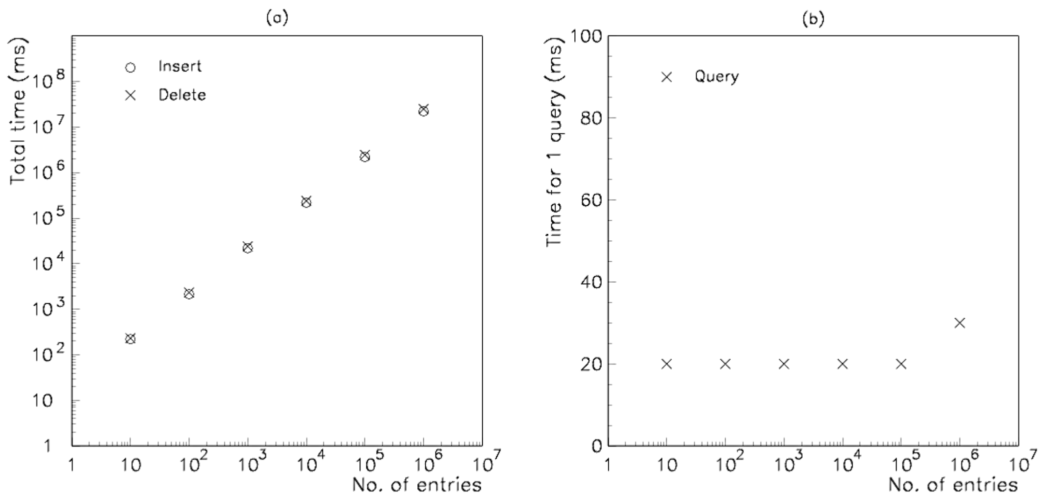

Figure 12: (a) Total time to add and delete mappings and (b) query the RMC using the C++ API.

Next, we analysed the query behaviour when there were multiple LFNs mapped to a single GUID (Figure 13). Whereas the insert/delete behaviour is constant, the query operations last longer the more LFNs exist for a single GUID. We used from 10 to 10,000 LFNs per GUID and observe that for larger numbers of LFNs the query time increases almost linearly.

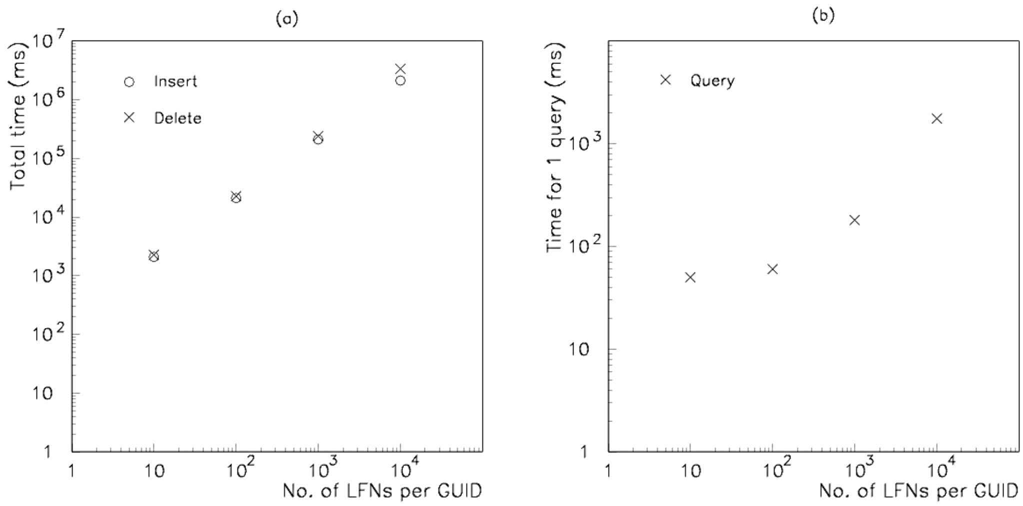

Figure 13: Total time to (a) insert and delete 10 GUIDs with varying number of LFNs, and (b) query for one LFN.

JAVA API

The performance of the Java API was measured in a similar way to the performance of the LRC Java API. A Java client class was written which used multiple threads to simulate many users concurrently using the RMC. It performs the required number of operations (create, query and delete) as quickly as possible using all the threads available. For each type of operation the average time to do the command was measured every 1000 operations.

Firstly, the time to insert a GUID:LFN mapping into the RMC was measured as the of number of entries in the catalog and the number of concurrent threads changed. Figure 14 shows how the average time to insert a mapping varies from using an empty RMC to using a RMC with 500,000 entries. The machines used for these measurements were lxshare0344 and lxshare0346 at CERN, and grid04 and grid06 at Glasgow.

As with the LRC, the shapes of the plots are very similar for the different numbers of concurrent threads, in that the insert time gradually increases with the number of entries in the catalog. As the catalog fills up, the insert time becomes more random and the data is more noisy.

Next, the total time to insert 500,000 mappings was measured for different numbers of concurrent threads. This effectively gives the throughput performance of the RMC.

Figure 15 shows that the throughput levels reaches a maximum at around 5-20 threads, a similar result as was seen with the LRC. The total time using 20 threads improved by around 40% compared to using a single thread. All the results for the RMC, as expected, are very similar to the single LRC results, since both services are almost identical in their implementation.

RMC COMMAND LINE INTERFACE

The command line interface is implemented in Java using the Java API. Table 5 shows some timing statistics showing the time to execute different parts of the command addAlias (in insecure mode) used to insert a GUID:LFN mapping into the RMC.

The timing results are very similar to the LRC command line interface results shown in section 5.2.2. The total time to execute the command was 3.1 seconds. The start-up script sets various options such

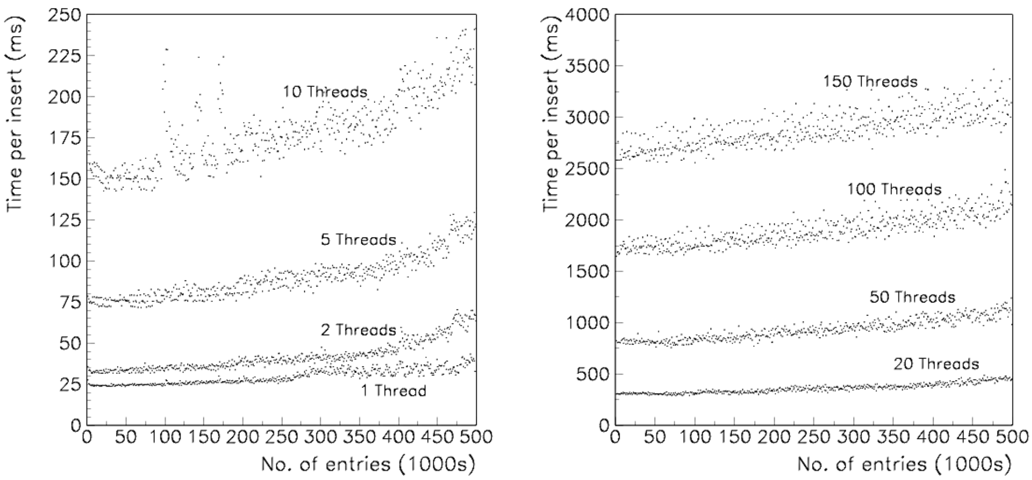

Figure 14: Time to insert one mapping into the RMC as the number of entries and number of concurrent threads varies.

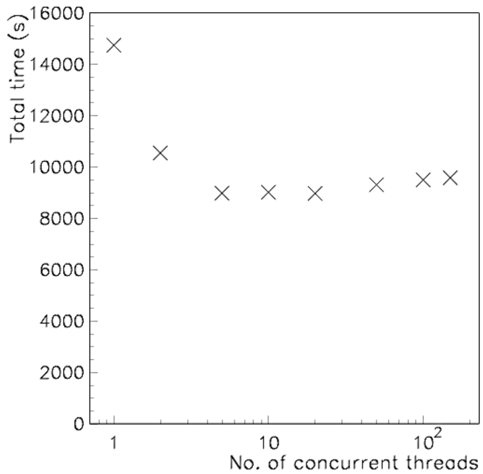

Figure 15: Total time to add 500,000 mappings using concurrent threads.

Time (s)	Operation
0 - 1.0	Start-up script and JVM start-up time
1.0 - 1.1	Parse command and options
1.1 - 2.1	Get RMC service locator
2.1 - 2.3	Get RMC object
2.3 - 3.0	1st call to the rmc.addMapping() method
3.0 - 3.1	2nd call to the rmc.addMapping() method
3.1	End

Table 5: Timing statistics for adding a GUID:LFN mapping in the RMC using the CLI.

as logging parameters and the class-path for the Java executable and this along with the time to start the Java Virtual Machine, took close to 1 second. After parsing the command line it took another 1 second to get the RMC service locator - during this time many external classes had to be loaded in.

The command was modified slightly so that it would insert two mappings in succession in order to show the effect of the class loading the first time the method is called. As with the LRC this has a large effect: the first call takes 0.7ms but the second time it takes only 0.1s. Thus for adding multiple entries to the catalog it is far quicker to use multiple calls to the Java or C++ APIs than the command line tool repeatedly, since every time the latter is used the overhead of dynamic class-loading is very large.

The same conclusions can be drawn as were drawn after the LRC command line interface test, that the RMC command line interface is about 2 orders of magnitude slower than the average time taken using the Java or C++ API, and so is not recommended for large scale operations.

5.3.3. SCALABILITY

The RMC is a simple service and thus scales very well for several LFN-GUID entries.

We have tested it successfully with 1,000,000 entries with a single LFN per GUID. In addition, we tested up to 10,000 LFNs per GUID.

5.4. REPLICA OPTIMIZATION SERVICE

Given that a set of replicas that are possibly spread all over the Grid, the Replica Optimization Service selects the best one with respect to network and storage access latencies. Within the EDG testbed, replica selection is only based on the network cost functions provided by WP7. However, a close collaboration with the EU CrossGrid project resulted in the integration of the Replica Optimization Service with the Data Access Estimation Service (DAES). On our WP2 testbed we performed benchmarks on a fully integrated optimisation service.

5.4.1. FUNCTIONALITY/INTEGRATION

Network-based replica selection: Together with WP7 we designed the API `getNetworkCost()` that returns the estimated network transfer time between various Grid sites. Within the EDG testbed, network monitoring is performed on the five major testbed sites, namely CERN, CNAF (Bologna), IN2P3 (Lyon), NIKHEF (Amsterdam) and RAL (Oxford). The estimation about the network transfer time is used by the Replica Optimization Service to locate the best replica. Experiments executed with WP7 showed that the estimated transfer times for files with sizes greater than or equal to 100 MB correspond well to the real transfer times. On average, we observed estimation errors lower than 15%, and for links between IN2P3 and NIKHEF less than 3%.

Storage access-based replica selection: Similar to `getNetworkCost()`, the Replica Optimization Service supports an API `getSECost()` that returns the estimated access times for files on a hierarchical storage system. This API is implemented by both WP5 of EDG and WP3 of CrossGrid. Currently, the API provided by WP5 returns only a binary value dependent upon whether a file is on disk or on tape. However, the implementation of CrossGrid returns more advanced access estimations. Results indicate that the estimated values correspond well to the real access times. Details can be found in [26]. In the following we only report on performance results of the CrossGrid implementation.

5.4.2. PERFORMANCE

In this section we measure the response time of the Replica Optimization Service for retrieving the best replicas.

JAVA API

getBestNetworkCost(): This method returns the network cost of the best replica given a specific logical file name (LFN). Internally, ROS contacts the RLS for retrieving all replicas of a given LFN. Next, ROS calls the network monitoring service with the SE-IDs and the file size of all replicas.

In our testbed setup, ROS is running on lxshare0313.cern.ch and the central database of the network monitoring service is running on ccwp71.in2p3.fr. Figure 16 shows the response time of ROS for calling `getBestNetworkCost()` with with an array of 1 to 128 replicas randomly selected from the 5 major testbed sites. Each call was repeated 10 times and the average response time is reported.

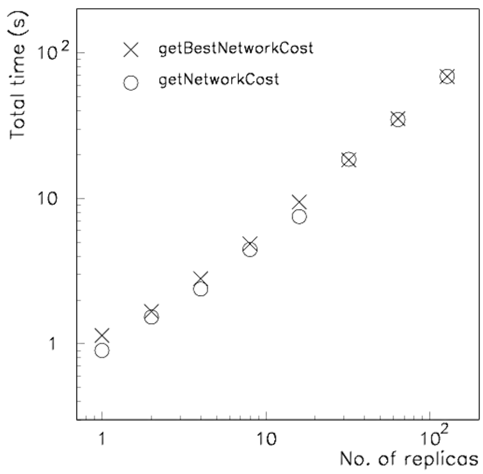

Figure 16: Total time to call getBestNetworkCost() and getNetworkCost() with different numbers of replicas.

It is important to note that the response time of the first call to `getNetworkCost()` is on average 0.8 seconds larger than for the successive calls. This is due to the time spent on dynamically loading the required Java libraries.

In order to see the overhead of the Replica Optimization Service, we compared these results with calling the network costs directly, i.e. without ROS. Again, each call was repeated 10 times and the average response time is reported. We can observe that the overhead of ROS is minimal.

In the next set of tests we measured the scalability of the network costs by calling ROS with multiple client threads. Since most of our benchmarks showed similar results, we only report on one specific setup.

The number of replicas was set to 16 and we called `getNetworkCost()` 32 times with various numbers of threads ranging from 1 to 32. The workload was evenly distributed among the threads running on the UI lxshare0344.cern.ch. For instance, in the case of one thread, the ROS client issued 32 calls; in the case of two threads, each ROS client issued 16 calls; finally, in the case of 32 threads, all 32 ROS clients issued 1 call each. We measured the time until all client threads received the results back from ROS.

We observed that the bottleneck of retrieving the costs of the best replica is not ROS but the central WP7 network monitoring archive. Thus, these tests basically evaluate the scalability of the network monitoring service. In Figure 17 we can see that the response time for the 32 calls is reduced by around 10% from 1 to 32 threads.

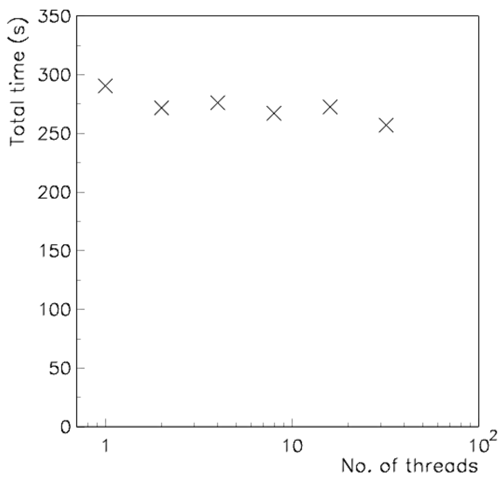

Figure 17: Total time to call getNetworkCost() 32 times with different numbers of concurrent threads.

getSECost(): This method returns the access costs of a set of files located on a specific hierarchical storage system.

In our testbed setup, ROS is running on lxshare0313.cern.ch and the Data Access Estimator of CrossGrid is running on pcrd24.cern.ch and on zeus04.cyf-kr.edu.pl in Cracow. Figure 18 shows the response times of ROS for calling `getSECost()` with various numbers of files ranging from 1 to 128. Each call is repeated 20 times and in each case the first file is assumed to be at pcrd24.cern.ch and the second one at zeus04.cyf-kr.edu.pl.

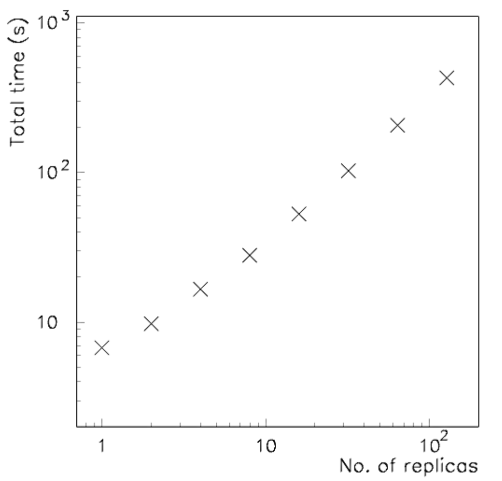

Figure 18: Total time to call getSECost() with different numbers of replicas.

We can see that the total response time varies linearly with the number of replicas. Similar to `getNetworkCost()`, the response time of the first call to `getSECost()` is on average 0.8 seconds larger than for the successive calls. This is due to the time spent on dynamically loading the required Java libraries. We also observed a slight difference of the response time between calling `getSECost()` on pcrd24.cern.ch (local to the UI machine) and on zeus04.cyf-kr.edu.pl (remote to the UI machine). The time difference is some 150 ms which is roughly two times the network round trip time between these machines.

Scalability measurements of `getSECost()` are part of future work since the Data Access Estimator is still being optimised.

COMMAND LINE INTERFACE

The command line interface is implemented in Java using the Java API. Table 6 shows some timing statistics indicating the time to execute different parts of the command `getNetworkCost()` (in insecure mode) used to get the cost of transferring a file from pcrd24.cern.ch to ccgridli07.in2p3.fr.

Time (s)	Operation
0 - 1.2	Start-up script and JVM start-up time
1.2 - 1.4	Parse command and options
1.4 - 2.7	Get ROS service locator
2.7 - 4.4	call getNetworkCost
4.4	End

Table 6: Timing statistics for calling getNetworkCost in the ROS using the CLI.

The total time to execute the command was about 4.4 seconds. The start-up script sets various options such as logging parameters and the class-path for the Java executable and this along with the time to start the Java Virtual Machine, took almost 1.2 seconds. After parsing the command line it took 1.3 seconds to get the ROS service locator. The time for calling `getNetworkCost()` is some 1.7 seconds which includes calling a remote service running in Lyon. Note that the time for calling `getNetworkCost()` drops from 1.7 seconds to 0.8 seconds for the second call (see Java API).

Table 7 shows the time for executing the command `getSECost()` for retrieving the access cost of a file stored at pcrd24.cern.ch. The total time for executing the command is 3.8 seconds which includes the time for calling the Access Estimation Service from CrossGrid.

Time (s)	Operation
0 - 1.2	Start-up script and JVM start-up time
1.2 - 1.4	Parse command and options
1.4 - 2.7	Get ROS service locator
2.7 - 3.8	call getSECost
3.8	End

Table 7: Timing statistics for calling getSECost in the ROS using the CLI.

5.5. REPLICA MANAGER

5.5.1. FUNCTIONALITY/INTEGRATION

The Replica Manager acts as the entry point to all the data management services. It uses the RLS and the RMC for cataloging data and the ROS for optimising file access. The Replica Manager provides four types of commands:

- **Management commands**: These commands combine data transfer and registration in the catalog services.
- **Catalog commands**: Interfaces to the catalogs. The basic functionality has been covered in the RLS section above.
- **Optimization commands**: Replica optimisation commands for optimised file locations and transfers.
- **File transfer commands**: Basic (third party) file transfer mechanism without updating the replica catalog.

Open issues:

- **Client tool only - no service:** In the current implementation, the Replica Manager is a client side tool only. This has several advantages like easy installation and configuration but on the other hand there is no server side processing like file check-summing, queued file transfers, checkpointing/restarting and rollback mechanisms. All these features should be added in the future in order to improve performance and reliability of the service.
- **Parallel queries:** Currently, all queries to the LRCs are done in sequence but should be done in parallel to improve performance.

5.5.2. PERFORMANCE

The Replica Manager is a client side tool that uses all of the services stated above (RLS, RMC, ROS) and thus depends on the performance and scalability of these services. Therefore the performance of the Replica Manager is closely related to the performance of these services and we refer to the previous sub-sections in this section for results. The next paragraph describes briefly the file transfer part of the Replica Manager. For further file transfer details and in particular optimisation results, we refer the reader to [18].

FILE TRANSFER COMMANDS

The Replica Manager uses primarily the GridFTP protocol and thus has similar performance results as a conventional GridFTP file transfer (see [14] for details). However, since the Replica Manager does additional existence checks and the Java command line tool has some significant overhead, the file transfer is slower by a constant factor as compared with a pure GridFTP file transfer using for example globus-url-copy.

5.5.3. REPLICA MANAGER SCALABILITY

Since the Replica Manager depends on the services it is using, scalability measurements need to take this into account and thus we refer to the catalog services for parts of the discussion.

However, for the end-user it is of major importance to know how many files can be transferred without errors using the Replica Manager. For this purpose we ran large scale tests which involved transferring 1,800 1GB files between NIKHEF (Netherlands) and CNAF (Italy) and registering them in the catalog using the Replica Manager command `copyAndRegisterFile`.

The 1.8 TB of data was successfully transferred in around 62 hours where each file transfer plus registration took between 1:45 and 1:49 minutes, depending on the current network throughput. The files were transferred using parallel streams. As compared to a single file transfer using `globus-url-copy` (1:35 min), the Replica Manager has a slight overhead due to the cataloguing and the file existence checks. This overhead can partially be overcome using the command `bulkCopyAndRegisterFile` where the additional startup time of the Java Virtual Machine and some class loading are eliminated since they are only needed for the first file transfer.

5.6. Security

5.6.1. Functionality/Integration

The two components of the Java Security Package, TrustManager and AuthorizationManager, have been extensively tested in order to assure the correct behaviour of their security functionality.

The TrustManager has been successfully integrated into WP2 and other work packages' (WP3, WP5) software for EDG v2.x, though experience of heavy usage has not been collected yet. The AuthorizationManager has not been integrated but initial tests have been carried out mainly internally within WP2. More feedback will be collected in the future.

5.6.2. Performance

Using the security modules creates a certain overhead due to the following operations: the creation of a secure communication channel between client and server, the authorisation of client requests according to a certain access control policy, etc. This overhead can be divided in two categories: unavoidable and implementation specific. The unavoidable part consists of SSL handshake and data encryption. The SSL handshake triples the network latency for the first network query-response pair. This happens because the SSL protocol requires at least two query-response exchanges with the service before the actual communication can start. Data encryption also slows things down because data is encrypted when sending and decrypted when receiving.

A comparison of performance with and without security is given in Table 8. It shows the time a Java client took to add a number of GUID:PFN mappings into a LRC using the Java API method addMapping(). There is a roughly constant overhead when using security due to the reasons discussed above. The Java clients in this test are relatively slow and turning on security obviously makes them slower. As already observed earlier, the Java clients pay some performance penalty because the Java virtual machine has to be started and external classes have to be loaded for each test.

Inserts	Secure client (s)	Insecure client (s)
1	0.77	0.07
10	7.07	0.54
100	55.44	3.38
1000	527.12	28.61

Table 8: LRC Java client performance comparison with security on and off

Some more results using security can be seen in the next section which contains Spitfire client performance tests and includes security tests.

One of the next steps to do is to improve the security modules to reduce the current performance penalty. In detail, as the Spitfire client performance results show, the secure C-client library is fast and a preliminary code analysis of the Java client has obtained several places for optimisation that should at least halve the performance penalty caused by the security in Java clients. However, security will always cause some additional performance overhead.

5.7. SPITFIRE

Here, we provide a short summary of Spitfire's functionality and performance. The functionality was tested and the performance was measured using stand-alone clients (Java and C) in secure and insecure modes. The results indicate that the database service as such (on the server side) is very fast, but that data transfers and Axis (Apache SOAP engine) add large overheads.

5.7.1. FUNCTIONALITY/INTEGRATION

Like many of the EDG software components, Spitfire is a web service that can be accessed with secure or insecure SOAP, i.e. XML messaging. In secure mode, the XML message headers can incorporate information about the user certificate being used to access a particular service. EDG Java Security's Tomcat add-ons take care of authentication and authorisation based on these certificates. The certificates can contain Virtual Organization Membership Service (VOMS) data, and VOMS authorisation has been tested in connection with Spitfire.

5.7.2. PERFORMANCE

The overheads of loading the client into memory and (in the case of a secure connection) the handshake with the server appear to be constant (about 2s with Java clients, less than 1s with C clients). The security overhead is about 3s with Java and less than 1s with C.

The server-side tasks consist of verifying the client, accessing the data and transforming the data into XML format. If the amount of data is large, the memory of the server will become congested since all data is held in memory during the process. In order to alleviate this problem, a PartialResultSet interface was developed. This interface allows the client to fetch only a part of the data at a time, and continue the process later. This mechanism does degrade the overall performance since a connection has to be re-established to fetch more data.

Rows	Query Method			
	1	2	3	4
10	0.007	13.2	8.2	1.2
100	0.026	14.3	10.0	1.8
1,000	0.6	31.2	26.9	11.8
10,000	6.3	223	207	192

Table 9: Spitfire client performance

Rows	server response time
10	0.31
100	0.37
1,000	0.40
10,000	2.07

Table 10: Spitfire server performance

CLIENT-SIDE PERFORMANCE

For the following client-side performance measurements, the queries of 10,000 rows of data use the PartialResultSets method. Each row in the data set consists of two `varchar` columns that had an average of 100 characters.

The query methods below are:

1. plain MySQL, no security, direct access to data
2. Java stand-alone client with security
3. Java stand-alone client without security and
4. C client with security.

The results, shown in Table 9, are given in seconds.

SERVER-SIDE PERFORMANCE

The server-side performance was measured by comparing a time stamp at the beginning and at the end of a request and calculating the difference. The request consists simply of issuing a query and generating its SpitfireResult (a data structure that contains the result of the query). The results below indicate that the result data structure is generated quickly and that the main overhead is that of Tomcat and Axis.

6. Simulation of Replica Optimisation

The development of the Replica Optimisation Service made use of the results from extensive simulations of various replica optimisation strategies. These were performed with the OptorSim simulation package, which was developed to mimic the structure of a real Data Grid and to study the effectiveness of different scheduling and replica optimisation algorithms within such an environment. Given (a) a Grid topology and resources, (b) a set of jobs that the Grid must execute and (c) an optimisation strategy, OptorSim simulates what would happen in the Grid if that optimisation strategy were in use. It provides us with a set of measurements in order to quantify the effectiveness of the strategy. OptorSim is written in Java™ and follows the architectural guidelines of all the WP2 services described in Section 4.

6.1. Architecture

The simulation is based on a simplification of the architecture of EDG and assumes that the Grid consists of several sites, each of which may provide computational and data-storage resources for submitted jobs (see Figure 19). Each *site* consists of zero or more *Computing Elements* and zero or more *Storage Elements*.

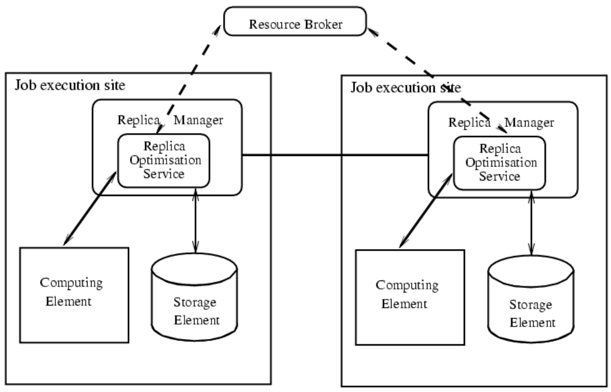

Figure 19: EDG Architecture.

Computing Elements run jobs, which use the data in files stored on Storage Elements. A *Resource Broker* controls the scheduling of jobs to Computing Elements, with the aim of improving the overall throughput of the Grid. In the simulation, sites without Storage or Computing Elements act as network nodes or routers. Grid sites are connected by *Network Links*, each of which has a certain bandwidth. A *Replica Manager* at each site manages the data flow between sites and interfaces between the computing and storage resources. The *Replica Optimisation Service* [20] inside the Replica Manager is responsible for replica selection, automatic creation and deletion of replicas.

Simulation Input. As input, OptorSim uses three configuration files. One file describes the *network topology* (network links between Grid sites and maximum available bandwidth for each defined link) and the components of each site (number of Computing and Storage Elements, as well as their sizes).

The second configuration file contains *information on the simulated jobs*, in particular the logical file names (LFNs) they need to access while executing. A file is characterised by its LFN and size. The order in which a job requests files is determined by the *Access Pattern* used [36]. The following access patterns are considered: *sequential* (all files are requested in a predetermined order), *random* (files are selected randomly from a set with a flat probability distribution), *unitary random walk* (set is ordered and successive file requests are exactly one element away from the previous file request, in a random direction), *Gaussian random walk* (as with unitary random walk, but files are selected from a Gaussian distribution centred on the previous file), and *Zipf* (file popularity is based on a Zipf-like distribution [39]).

Another important aspect modelled in OptorSim is background (i.e. non-Grid) network traffic, which use a sizable proportion of the underlying network resources and can vary unpredictably over time. The third input configuration file contains *information about background network traffic*, which is based on network monitoring data between Grid sites.

6.2. Optimisation in OptorSim

OptorSim performs two-stage optimisation: *scheduling optimisation* (deciding where a job should be executed) and then *run-time optimisation* (deciding which is the best replica for a file request and how best to position the data).

The Resource Broker uses a scheduling optimisation algorithm to calculate the cost of running a job on a group of candidate sites and submits the job to the site with the minimum estimated cost. A run-time optimisation algorithm is used by the *Replica Optimisation Services* (or *Optimisers*) to locate the best replicas. The main advantage of two-stage optimisation is that scheduling decisions are based on both the location of data and the status of network links between Grid sites, while (re)optimisation during the run-time of a job takes into account dynamic variations in the distribution of data and in the behaviour of network resources.

6.2.1. Scheduling Optimisation

The job scheduling algorithms used by the Resource Broker are based on one or more of the following cost metrics [38]:

- *Access Cost*. The estimated cost, based on the current network status, for obtaining all the files required by a job. This metric uses the Replica Catalogue to look up all the replicas for each required file. The access time for each replica is calculated and thus the best replica can be found for each file. The combined access time for the best replicas is used to rank candidate sites.
- *Queue Size*. The number of jobs waiting in the queue at the candidate site. We assume that only one job at a time can run on each CE.
- *Queue Access Cost*. For each job in the queue the access cost is calculated as for the *Access Cost* algorithm. The access costs for all jobs are summed to give a total estimated access cost for all the jobs in the queue.

6.2.2. Run-time Optimisation

When a file is requested by a job, the local Optimiser executes the following tasks [38]:

- *Replication Decision*. If a requested file is not present on the site's SE, this process decides whether local replication of this file should take place. If the Optimiser decides not to replicate a file then the job must access that file remotely.
- *Replica Selection*. Whenever remote replicas need to be accessed, this process selects the best of those available. In general, the selection criterion depends on the chosen evaluation measure.

- *File Replacement*. When a remote replica has been selected for replication to the local SE, the SE might not have sufficient spare capacity. In this case, one or more local replicas must be deleted. The selection criteria for deciding which locally stored replicas to delete depend on some estimate of future usefulness.

A specific combination of algorithms for each stage defines a run-time replica optimisation strategy. A number of specific optimisation strategies have been implemented in *OptorSim*. Here we report on three of them: one based on the traditional *LFU (Least Frequently Used)* algorithm, and two economic strategies.

The LFU-based strategy will always replicate files to storage local to the Computing Element on which the job is running. Replica Selection is achieved by querying a Replica Catalogue to locate all replicas. The replica that can be accessed in the shortest time, under the current network conditions, is chosen. If the local storage is full, the file that has been accessed the fewest times in the previous time window is deleted, creating space for the new replica.

The two economy-based strategies are similar to each other, but use two different prediction functions, one binomial-based and the other Zipf-based, to calculate file values used in the replication and file replacement decisions. If the potential replica under consideration has a higher value than the lowest-valued file currently in the local storage, that file is deleted and the new replica is "bought". If local storage is not yet full, the economic models will always replicate.

The file value is approximated by the number of times it is expected to be accessed in a future time window $\delta T'$, based on the file access history for the previous time window δT. The binomial prediction function constructs a binomial distribution of file popularity, centred on the mean number of file accesses in δT. The value of the file in question is then found by checking where it lies on that distribution. The Zipf prediction function orders the files into a Zipf distribution according to their popularity in δT, and takes the value from there.

Replica Selection in the economic models is based on the auction protocol for buying and selling files described in [37].

6.3. Evaluation of Optimisation Algorithms

We present simulation results, extracted from [38], obtained by using OptorSim to evaluate and compare the optimisation strategies for both scheduling and run-time optimisation. We considered the following measures:

Mean Job Execution Time The mean job execution time is defined as the total time to execute all the jobs, divided by the number of jobs completed.

Network Usage We define *effective network usage* r_{ENU}:

$$r_{\text{ENU}} = \frac{N_{\text{remote file accesses}} + N_{\text{file replications}}}{N_{\text{local file accesses}}},$$

where $N_{\text{remote file accesses}}$ is the number of times the CE reads a file from a SE on a different site, $N_{\text{file replications}}$ is the total number of file replications, and $N_{\text{local file accesses}}$ is the number of times a CE reads a file from a SE on the same site (we assume infinite bandwidth within a site).

For a given network topology, a lower value of r_{ENU} indicates the optimisation strategy is better at replicating files to the correct location.

CE Usage Another resource which is of interest is the computational power usage, which we define as the percentage of time that a CE is running jobs. Henceforth, we use the term CE usage, which is the total computational power usage for all the CEs on the Grid. A good scheduler should be able to maximise the CE usage by spreading the workload, avoiding the situation where some sites lie idle while others have long queues of jobs.

6.4. Simulation Results

Access Patterns and Scheduling. We start our evaluation by studying the impact of the scheduling algorithm used by the Resource Broker on a given run-time optimisation strategy. The scheduling algorithms presented above are analysed and in addition, we consider an algorithm that schedules to a random CE. For each scheduling algorithm we consider some of the access patterns described previously i.e., *sequential*, *Gaussian random walk* and *Zipf*.

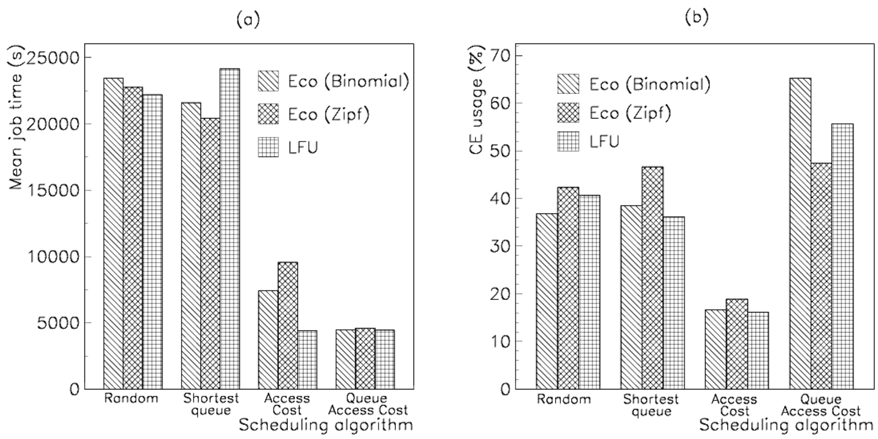

Figure 20: (a) Mean job time and (b) CE usage for various optimisation algorithms and sequential access pattern.

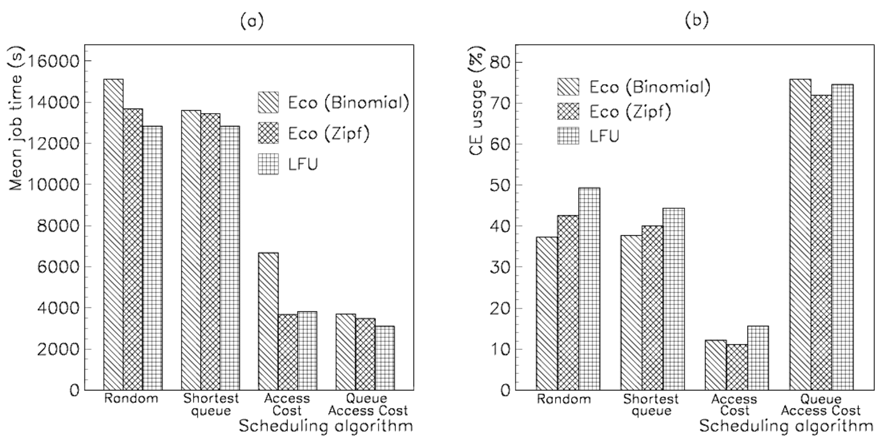

Figure 21: (a) Mean job time and (b) CE usage for various optimisation algorithms and Gaussian random walk access pattern.

Results showing the mean job time and CE usage for the three optimisation strategies and the three access patterns are shown in Figures 20, 21, and 22.

The *Random* and *Shortest Queue* algorithms show similar performance and generally have the longest

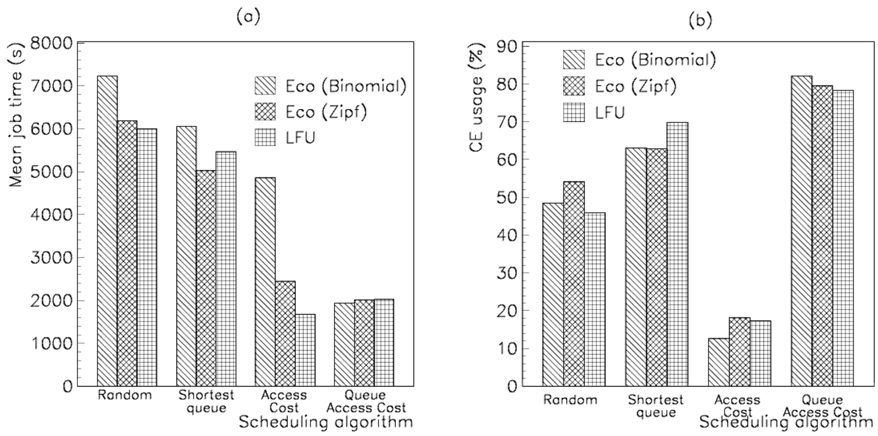

Figure 22: (a) Mean job time and (b) CE usage for various optimisation algorithms and Zipf access pattern.

mean job times. The *Access Cost* algorithm has a lower mean job time but has the lowest CE usage, due to the fact that jobs are only scheduled to sites with high network connectivity.

The mean job time is lowest and CE usage is highest when we use the *Queue Access Cost* algorithm. This gives the best balance between scheduling jobs close to the data while ensuring sites with high network connectivity are not overloaded and sites with poor connectivity are not idle.

Scalability of Optimisation Algorithms. In the next set of tests we study the scalability of the optimisation algorithms by varying the number of jobs from 100 to 1,000 to 10,000. The effective network

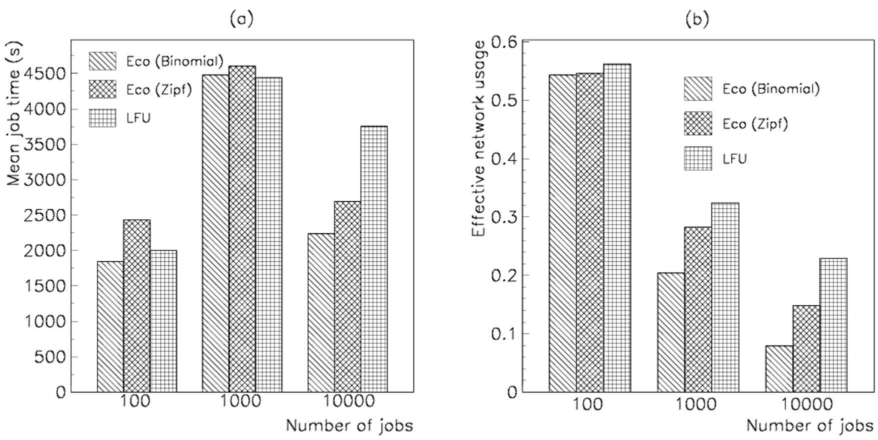

Figure 23: (a) Mean job time and (b) effective network usage for different number of submitted jobs.

usage, shown in Figure 23(b), decreases with the number of jobs submitted, as we might expect, since the access histories used by the optimisation strategies to make replication decisions take time to build up and stabilise. The main advantage of the economic strategies is that they use up considerably less network bandwidth than the LFU strategy.

This scalability can also be seen in the mean job times (Figure 23(a)), with the economic strategies becoming more effective with an increased job load.

Effects of non-Grid Network Traffic. In all the previous evaluations we included non-Grid background traffic in our network model. Here, we examine the effect this has on Grid performance by comparing results with and without the inclusion of background (see Figure 24). As would be expected,

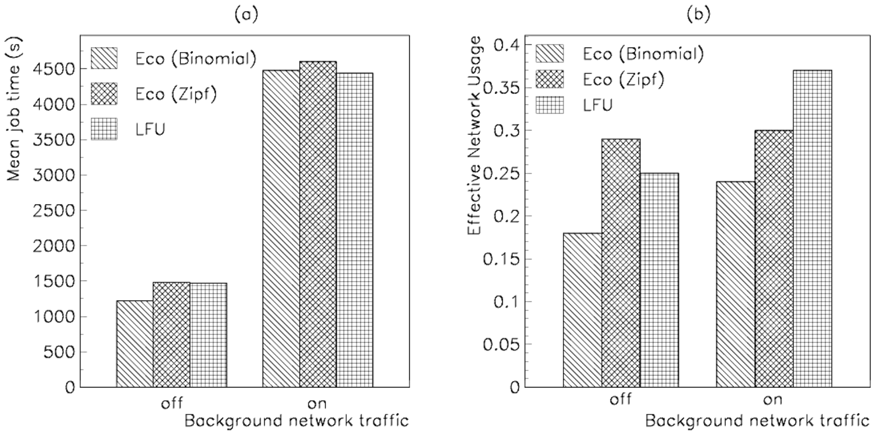

Figure 24: Effects of background network traffic on (a) mean job time and (b) effective network usage

there is a large increase in mean job time when we simulate the background network traffic. There is also a big increase in the effective network usage for the binomial-based economic strategy and LFU strategy, while for the Zipf-based economic strategy it remains roughly constant. This is perhaps due to the changing network bandwidths leading to less reliable replication decisions by the optimisation strategies, which in the long term means that more replication takes place - except in the Zipf-based economic strategy, which seems to be the most stable to fluctuations in the network bandwidth.

6.5. CONCLUSION

We have shown that the choice of strategies used can affect both the extent to which Grid resources are exploited and the throughput of Grid jobs. In particular, our simulations show that *Queue Access Cost*, a scheduling algorithm which takes into account both the file access cost of jobs and the workload of computing resources, is the most effective at optimising computing and storage resources and reducing the average time to execute jobs.

We have also proven that a suitable choice of data replication strategy can improve Grid performance; for most situations, particularly with large numbers of jobs, the economy-based strategies we have developed have the greatest effect, regardless of the presence of background (non-Grid) network traffic.

7. Conclusion, Open Issues and Future Work

Most of the services and modules provided by WP2 have satisfied the basic user requirements and thus software system can be used efficiently in the DataGrid environment and partly even beyond. However, several services still needs some particular work.

This section gives an overview of additional issues and specific work in the area of data management that could not be carried out within the time-frame of the EDG project. These issues need to be addressed in order to augment the functionality of the WP2 middleware as well as to improve the robustness and usability.

7.1. Limitations of WP2 Components

7.1.1. Replication Services

In their current state, the replication services of WP2 are not complete as we have proposed them according to our initial design. The shaded components in Figure 1 represent those components that were included in the original design but were not implemented primarily due to time constraints. These remaining components are needed to provide a functionally complete set of replication services for data management on the Grid. This section deals with each of these 'missing' services and functionalities in detail. They should all be addressed in some manner in future projects, so these issues are valid also outside the strict scope of EDG-WP2.

Processing

Pre- and post-processing hooks were foreseen to be available through the Replica Manager. The reason to have such hooks is that many Virtual Organizations have use-cases concerning replication where they have to add some processing before the data can be replicated. Examples are

- checksums
- specialized validation after copy
- encryption and decryption of data
- additional entries to be made in application catalogs
- actual data generation from templates

The processing interface needs to be thought through carefully to be able to meet all the various needs of different application groups and at the same time be simple enough to easily integrate with the existing components.

Replica Initiation and Access History

In the current optimization service replication occurs only when a user requests a file and it is not available on the close SE of the CE on which the job is running. A complete optimization service should be able to control when and where to replicate files, independent of users' actions. Such replication would effectively try to create an optimal distribution of replicas around the Grid.

There are several interesting models of optimization techniques that should be investigated, especially methods of dynamic replication based on usage access history, such as those developed in OptorSim. If a file is accessed heavily through remote access methods at a given site, it should be replicated to that given site so that a local copy is available. There are many possibilities on how such an optimization can be achieved, some of which have successfully been tested in commercial peer-to-peer applications. See also Section 7.2. below.

REPLICA CONSISTENCY

In a distributed system like the Grid a lot of things can go wrong. Network connections are severed, computers go up and down, so even for very well-written, robust middleware we have to allow a margin of error. One of the most serious problems that could result in a disfunctional Grid system is that the information about the data locations in the replica catalog become inconsistent with the location of the data themselves.

To find and correct inconsistencies between catalogs and storage systems there was a plan for a consistency service that would periodically sweep through the catalogs checking whether the catalog entries and the files on storage are consistent.

Currently, such checks are performed manually by the system administrators, a very time consuming, erroneous and tedious task. A replica consistency service should be a high priority to ensure a reliable and functioning Grid system. Preliminary results on a Consistency Service are reported in [15].

COLLECTIONS

Datasets or more specifically file collections are not implemented in the final set of WP2 data management services. There are many different types of collections that can be defined:

Free logical collections These are defined only by a list of LFNs. The file replicas themselves can be anywhere on the Grid. A collection may or may not contain another collection, depending on the definition and the desired semantics.

Confined logical collections These are also defined as a list of LFNs, but with the extra constraint that the files in the collection are always available on the same storage element or site. When the collection is replicated, all of its constituent files must be copied to the destination.

Metadata defined collection A collection of LFNs that are defined wholly by a metadata query on some external metadata. This may either be fixed at the time of the collection's creation or it may be dynamic, in which case, the size of the collection grows as new files which fulfil the query are added to the Grid.

Confined physical collections Similar to the type used in GDMP, these are definitions of a set of physical file-names (e.g. a filename glob in a given directory). Physical file-sets are typically replicated together from one site to another, with a corresponding update in the replica catalogs.

The semantics and integrity of different types of collections under various operations (e.g. deletion, replication, collection membership addition/deletion) needs to be examined more carefully, particularly with regard to access control lists and quotas.

The easiest policy to implement is that of no control at all; i.e. although a collection may have an owner, the existence of that collection does not preclude the deletion of some of its member files. More complex policies have issue when quotas are involved, where the existence of one person's confined collection could 'pin' the existence of thousands of files that are using another user's quota. On the other hand, if a simple "no-control" policy is used, allowing deletion of some of a collection's member files from the

Grid can lead to a 'hanging symlink' problem. The various policies to handle these sorts of conflicts need to be examined carefully.

Another example is the behaviour of nested collections under operations such as delete. For example, are nested deletes of all nested collections ever desirable? Again, the hard issues only really begin to surface when access control lists and user quotas are considered.

Collections can be defined in a number of ways, each with different semantics, dependent upon the operations being performed on them and upon various policies. Each use-case for collections should be carefully examined to understand which type of collection is appropriate and what semantics and policy decisions are suitable for it.

REPLICATION SESSIONS

There are use cases, especially from the HEP community, where the replication of a single file or even of a collection of files is not seen as an atomic operation - several sets of (collections of) files, spawned across several jobs is, involving a lot of different operations (i.e. any of replication, registration, unregistration, deletion, etc).

This can be managed in a straightforward manner if replication jobs are assigned to a session. The Session manager would hand out session IDs and finalize sessions when they are closed, i.e. only at that time would all changes to the catalogs be visible to all other sessions. In this context sessions are not to be misinterpreted as transactions, as transactions may not span different client processes; sessions are also managed in a much more lazy fashion. The details of how the APIs would need to be modified and the session manager be integrated with all replication tools would need to be worked out in detail.

REPLICA SUBSCRIPTION

In the first version of the EDG testbed WP2 provided a replica subscription facility, GDMP. The intent was to replace GDMP with a more robust and versatile facility for testbed 2 but this service was not developed due to time pressures in the project (the stability of the other services was higher priority). The functionality to automatically distribute files once they are available is still a much-needed one and will need to be addressed in systems using the EDG WP2 tools as a high priority.

7.1.2. METADATA SERVICES

Currently, the metadata support in the RMC is limited to of O(10) basic-typed attributes, which can be used to select sets of LFNs. The RMC cannot support many more metadata attributes or more complex metadata structures. More complex metadata structures can be exposed and queried via the Spitfire tool with full Grid authentication and configurable authorization.

There is ongoing work in the context of the GGF OGSA-DAI working group to define proper interfaces for data access and integration, much of their findings can be used to refine and re-define the metadata structures of Spitfire and the RMC (see also the section on OGSA below).

7.1.3. SECURITY

WP2 has contributed substantially to the design of the security framework of EDG. The edg-java-security package has been integrated with all EDG web services based on the Tomcat container (from WP2, WP3 and WP5). There is additional work necessary to improve the performance of the security components and there is still a lot of room for improvement for the configuration tools for the AuthorizationManager.

The introduction of standard languages (XACML, SAML) for the definition of access control policies is also a direction where future work could be invested.

As for open issues; in order to have full functionality in the security domain, not just for the data management services, work has to be invested in the following areas:

Delegation of Rights Currently we are limited by the lack of proper delegation mechanisms to provide pure client-server services. In the Grid, one of the most powerful concepts is one of a 'higher level' service which consolidates the specific features of underlying services and presents a 'new' service to the users with easier to use, well-managed interfaces. An example would be a replica manager *service* - currently there is only a replica manager client. The service could coordinate concurrent replications initiated from a given site. Currently multiple jobs may issue the same replication request, resulting in multiple remote file transfers since no coordination exists between such requests. WP2 has started work on a prototype of a delegation system which could be taken as a starting point for further activities in this area.

Ticket signing/action delegation system Instead of full delegation, this simple delegation system is an interesting alternative.

Encryption of Data Especially for biomedical data where the content is sensitive it is important to provide data encryption facilities to protect the data content from unauthorized access. Many existing encryption techniques could be leveraged to offer this additional functionality.

Mutual Authorization Currently, authorization is enforced at the server-end, where the system authorizes the user to access some resource. There are plans to provide mutual authorization, enabling also the client-end to check that the server is authorised to perform the requested service. Mutual authorization is especially important in the health care field where medical data must be stored only on trusted servers.

Access Policy Management The authorization mechanisms provided by EDG are straightforward (one might also say simplistic) in their usage. More elaborate administration can be provided through access policies. There are many existing solutions today as well, but none have been chosen nor tested thoroughly enough in a Grid context to date.

Online Credential Repositories (OCR) Since people are transient creatures, it would be wrong to assume that they always use the same user interface to access the Grid. OCRs or other mechanisms for single sign-on such as Liberty Alliance, `www.projectliberty.org` , would ease this aspect of secure human-interactions with the Grid.

Centralized Policy Configuration A tool providing policy configuration and policy propagation to configure site services centrally would ease the life of site administrators tremendously.

7.2. OPTIMISATION STRATEGIES

The current replica optimisation component of the Grid data management software (Optor) is designed to maximise the efficient usage of the available networking and storage resources upon the Grid. However, the current software is conservative in terms of functionality, and there has been much recent research in this area that should be leveraged to improve the capabilities and the performance of the optimisation systems. See also the description of the Replica Initiation and Access History components for Replication Services 7.1.1.

The **OptorSim** simulation could be improved to develop better algorithms for Grid optimisation. Areas for improvement are:

- *Validation of the simulation results against file access patterns from the applications.* These file access patterns should be measured across a variety of experimental use-cases.
- *More realistic simulation of the computing element clusters.*
- *More realistic simulation of the mass storage systems*, to investigate the impact of file staging on a site's efficiency and upon the overall Grid efficiency.
- *Continue to use the simulation to investigate the effect of background network traffic.* The background traffic (and background non-Grid jobs) have an influence on the overall efficiency of the Grid data management systems, including the sharing of resources among different Grid enabled application groups.
- *Investigate the effect of various Grid failure modes.* These include network loss, data loss and job mis-scheduling.

7.3. Common Services

One of the experiences gained in EDG is the need for common, robust and well tested components that can be reused for common middleware tasks. Having single implementations of common low-level middleware code reduces the manpower effort required to maintain the quality of these components and also reduces the number of possible areas for interoperability difficulties.

Areas for common code development are:

User Interfaces. Common command-line and graphical user interfaces, as well as web-based tools and portals could benefit from a uniform look and feel and from a well-tested underlying implementation.

Logging Frameworks. Such that all components within the project use the same logging standards and formats.

Error Reporting. The uniform handling of errors, definitions of error codes and descriptions can greatly benefit not just the end-user but also the Grid middleware developer.

Policy Management. Standard and configurable mechanisms for what to do in unforeseen situations. The most common situation is the one where a requested Grid resource is down. For example the replica manager's policy on how to react if one if the catalogs is unavailable could be configurable and managed through default policy manager tools.

Some of these common components are available at a lower cost than others, e.g. J2EE containers maintain a variety of common services that any service running in the container can make use of; other services would need to provide their own.

The OGSI definitions also provide some directions on common usage and tools for Grid services 7.4..

7.4. OGSA / OGSI

The Open Grid Service Architecture is a GGF standard for Grid service architectures which define in generic terms the structure and mechanisms that have to be made available by Grid services in order to be 'OGSA compliant'. The Open Grid Services Infrastructure (OGSI) working group has released the first version of a document that describes in great detail what these interfaces should look like [32].

All of the WP2 services can be made to adhere to the OGSI standard with some effort. In addition, the security components would need to be modified in order to be compliant with the OGSA Web Services Security specification.

In the area of metadata management, WP2 has participated in the shaping of the DAIS standard [21], specifying data access and integration interfaces not just to files but also to databases and objects. There is still a lot of work to be done also in terms of prototyping of new ideas to eventually achieve a fully functional Grid metadata system.